DOCUMENT RESUME

ED 235 779 IR 010 852

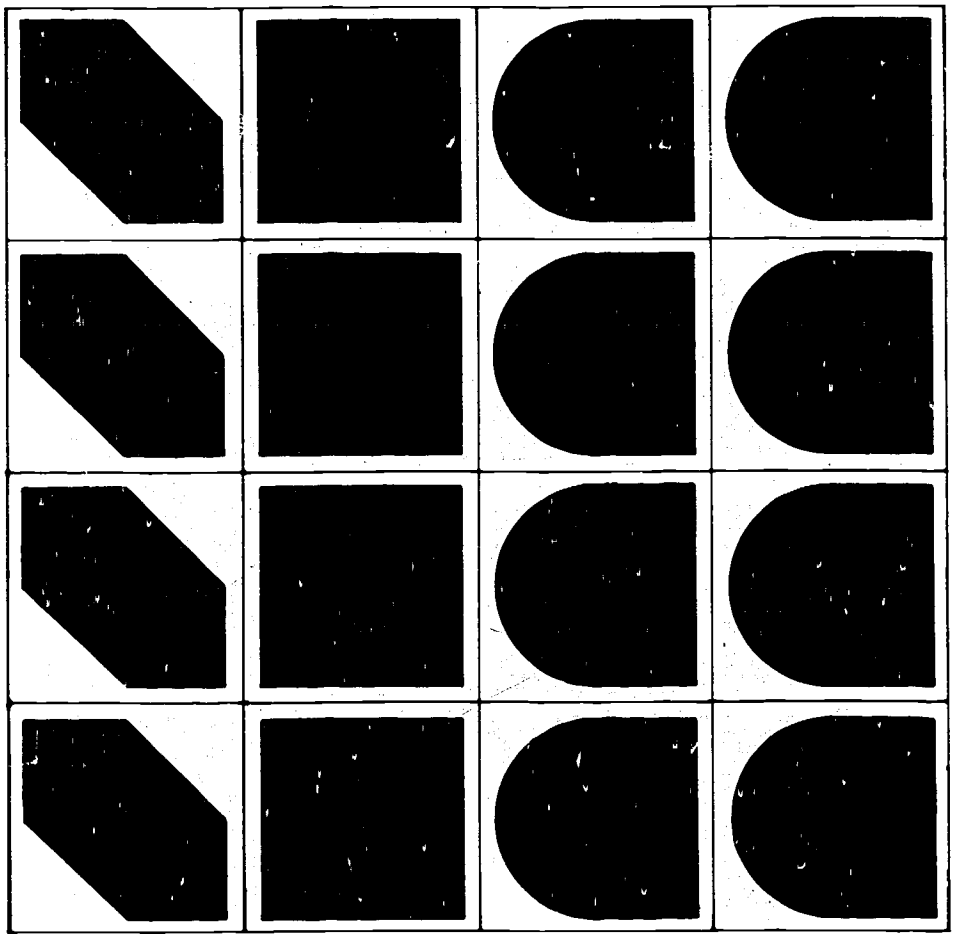
AUTHOR Bonnette, Della, Ed.
TITLE Proceedings of the NECC/5 National Educational Computing Conference 1983 (5th, Baltimore, Maryland, June 6-8, 1983).
REPORT NO ISBN-0-8186-0050-0
PUB DATE Jun 83
NOTE 422p.; Published by the IEEE Computer Society Press.
AVAILABLE FROM Professor Ted Sjoerdsma, University of Iowa, Computer Science Division, Iowa City, Iowa 52242 ($15.00).
PUB TYPE Collected Works - Conference Proceedings (021) -- Viewpoints (120) -- Reports - Descriptive (141)

EDRS PRICE MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS *Computer Assisted Instruction; *Computer Literacy; Computer Managed Instruction; *Computer Programs; *Computer Science Education; Instructional Materials; Material Development; Programing; *Programing Languages; Science Education; Teacher Education
IDENTIFIERS *Computer Uses in Education; LOGO Programing Language

ABSTRACT

Recent research and current trends in the field of computers and education are reflected in this collection of reviewed papers, tutorials, panels, project presentations and other sessions. More than 80 papers are grouped in the following topic areas: administrative applications, composition and literature, computing for the learning disabled or handicapped, computer services, computer science curricula, computers in education, computing in the non-curricular support role, pre-college computer science, computer science software, LOGO, alternative approaches to providing computing facilities, computer uses in education, science, computer literacy, computer education for elementary school teachers, commerce, computer science--teaching programming, computers in science education, computer assisted instruction, computers in education at an early education level, computer education for secondary school teachers, mathematical needs of computer sciences, computer-based education, teacher training, pre-college instructional uses of computers, mathematics and statistics, courseware development and evaluation, and pre-college computer services. An additional 39 papers are included under the headings of tutorials, invited sessions, and special sessions. (LMM)

Proceedings of NECC/5
National Educational Computing Conference 1983



CONFERENCE: June 6-8, 1983, Baltimore, Maryland
HOST: Towson State University, Baltimore, Maryland

ISBN 0-8186-0050-0
IEEE CATALOG NO. 83CH1888-7
LIBRARY OF CONGRESS NO. 83-80814
IEEE COMPUTER SOCIETY NO. 490

IEEE COMPUTER SOCIETY PRESS

EDITED BY

Della Bonnette
University of Southwestern Louisiana
Lafayette, Louisiana

Published by IEEE Computer Society Press
1109 Spring Street
Suite 300
Silver Spring, MD 20910

ISBN 0-8186-0050-0 (Paper)
ISBN 0-8186-0051-9 (Casebound)
ISBN 0-8186-0052-7 (Microfiche)

National Educational Computing Conference
June 1983

Cover designed by Madeline Windauer

NECC STEERING COMMITTEE

Ronald Anderson
University of Minnesota

Alfred Bork
University of California, Irvine

Nell Dale
University of Texas at Austin

Karen Duncan
Health Information Systems

Francis Edwards
Towson State University

Gerald L. Engel
Christopher Newport College

Mary Dee Harris Fosberg
Loyola University

John Hamblen
University of Missouri-Rolla

Diana Harris
University of Iowa

Harry Hedges
Michigan State University

Paul Heller
EDUCOM/EDUNET

Lawrence Jehn
University of Dayton

Sister Mary Kenneth Keller
Clarke College

Jesse C. Lewis
Jackson State University

Doris Lidtke
Towson State University

James Lubkin
Michigan State University

Mike Mulder
Servio Logic Corporation

Richard Pogue
Medical College of Georgia

Joseph Raben
Queens College/CUNY

Nancy Roberts
Lesley College

Alan L. Roecks
San Antonio ESC

Jean Rogers
University of Oregon

William Ryan
Swarthmore College

Ted Sjoerdsma
University of Iowa

Dennis Spuck
University of Houston

E. M. Stanan
University of Missouri-Columbia

David Stonehill
University of Rochester

Joe Turner
Clemson University

NECC Conference Committee

General Chairperson	Doris K. Lidtke Towson State University
Vice-Chairperson	Robert Caret Towson State University
Program Committee	Joseph Turner Clemson University
Contributed Papers	Joseph Turner Clemson University
Society Sessions and Panels	Jean Rogers University of Oregon
Project Presentations	William Ryan Swarthmore College
Special Sessions	William Dorn University of Denver
Films	Lillian Cassel Goldey Beacom College
Birds-of-a-Feather	David Stonehill University of Rochester
Session Chairpersons	Patricia Powers Goucher College
Workshops	Ralph Lee University of Missouri-Rolla
Proceedings	Della Bonnette University of Southwestern Louisiana
Publicity	James Adams Association for Computing Machinery
Exhibits	Gerald Leach-Lewis IEEE/Computer Society
Evaluation of NECC/82	Alan Roecks Education Service Center-San Antonio
Mailings	Carol Edwards Towson State University
Local Arrangements	Francis Edwards Towson State University
Co-chairpersons	Michael Haney Towson State University
	Charles Parrish Towson State University
Information Desk	Clarence Miller Maryland State Department of Education
Continuing Education Units	Robert Wall Towson State University
Social Events	Joyce Currie Little Towson State University
	Dick Austing University of Maryland
Restaurants	Ann Wagner Towson State University
Student Assistants	TSU ACM Student Chapter and others
Press Relations	Gerald Riggleman Towson State University

EXHIBITORS

ADDISON-WESLEY PUBL CO, READING, MASSACHUSETTS
ASSOCIATION FOR COMPUTING MACHINERY, NEW YORK
ANAHEIM PUBLISHING CO, FULLERTON, CALIFORNIA
APPLE COMPUTER, CUPERTINO, CALIFORNIA
ARDS, WASHINGTON, D.C.
ATARI, INC, SUNNYVALE, CALIFORNIA
BOBBS-MERRILL EDUC PUBL CO, INDIANAPOLIS, INDIANA
BOYD & FRASER PUBLISHING CO, SAN FRANCISCO, CA
C & C SOFTWARE, WICHITA, KANSAS
CHARLES E.MERRILL PUBLISHING, COLUMBUS, OHIO
CLASSROOM COMPUTER NEWS
COMMODORE BUSINESS MACHINES, WAYNE, PENNSYLVANIA
COMPUTER SCIENCE PRESS, ROCKVILLE, MARYLAND
COMPRESS-SCIENCE BOOKS INT'L, WENTWORTH, N.H.
CONDUIT(UNIV OF IOWA), IOWA CITY, IOWA
THE CONTINENTAL PRESS, ELIZABETHTOWN, PA
CORONADO PUBLISHERS, NEW YORK CITY
DEVELOPMENTAL LEARNING MATERIALS, ALLEN, TEXAS
DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS
DILITHIUM PRESS, BEAVERTON, OREGON
EDUCATIONAL TESTING SERVICE, PRINCETON, N.J.
EDUCOM/EDUNET, PRINCETON, NEW JERSEY
EDWARD ARNOLD, BALTIMORE, MARYLAND
ENTELEK, PORTSMOUTH, NEW HAMPSHIRE
EPIE-EDUCATIONAL PRODUCTS, NEW YORK CITY, N.Y.
FOLLETT LIBRARY BOOK CO, CRYSTAL LAKE, ILLINOIS
GAMCO INDUSTRIES,INC, BIG SPRING, TEXAS
GLOUCESTER COMPUTER,INC, GLOUCESTER, MASS
GOUCHER COLLEGE, TOWSON, MARYLAND
GREGG/McGRAW-HILL, NEW YORK CITY, N.Y.
HARPER & ROW PUBLISHERS,INC, NEW YORK CITY, N.Y.
HARTLEY COURSEWARE,INC, DIMONDALE, MICHIGAN
HOLT,RINEHART & WINSTON, NEW YORK CITY, N.Y.
HOUGHTON MIFFLIN CO, BOSTON, MASSACHUSETTS
IBM CORPORATION, BETHESDA, MARYLAND
INFORMATION SYNERGY,INC, PRINCETON, NEW JERSEY
J.L.HAMMETT COMPANY, BRAINTREE, MASSACHUSETTS
JOURNAL OF COMPUTERS IN MATHS & SCIENCE,
K-12 MICROMEDIA, WOODCLIFF LAKE, N.J.
LAWRENCEVILLE PRESS, LAWRENCEVILLE, N.J.
LITTLE, BROWN & CO, BOSTON, MASSACHUSETTS
BROOKS/COLE PUBLISHING CO, MONTEREY, CALIFORNIA
McGRAW-HILL PUBLISHING CO, NEW YORK CITY
MILLIKEN PUBLISHING CO, ST.LOUIS, MISSOURI
MONROE SYSTEMS FOR BUSINESS, MORRIS PLAINS, N.J.
RADIO SHACK, FORT WORTH, TEXAS
RANDOM HOUSE, NEW YORK CITY
RESTON PUBLISHING COMPANY, RESTON, VIRGINIA
SCOTT INSTRUMENTS, DENTON, TEXAS
SCHOLASTIC,INC, NEW YORK CITY, N.Y.
SPRINGER-VERLAG,N.Y. NEW YORK CITY
STERLING SWIFT PUBLISHING, AUSTIN, TEXAS
SUNBURST COMMUNICATIONS, PLEASANTVILLE, N.Y.
SYNTAURI CORPORATION, PALO ALTO, CALIFORNIA
SYSTEMS DESIGN ASSOCIATES, CHARLESTON, W.VA
TECHNICO,INC, BALTIMORE, MARYLAND
TERRAPIN,INC CAMBRIDGE, MASSACHUSETTS
WADSWORTH PUBLISHING CO, BELMONT, CALIFORNIA
WADSWORTH PUBLISHING CO (INT'L DIVISION)
WEST PUBLISHING CO, SANTA CLARA, CALIFORNIA
WEBSTER/McGRAW-HILL, NEW YORK CITY, N.Y.
JOHN WILEY & SONS, INC, NEW YORK CITY, N.Y.
QUEUE,INC, FAIRFIELD, CONNECTICUT
ZERO PAGE,INC, COLORADO SPRINGS, COLORADO

COOPERATING SOCIETIES

The Conference is hosted by Towson State University in cooperation with:

American Association for Medical Systems and Informatics

Association for Computing Machinery Special Interest Groups on:
- Computers and Society (SIGCAS)
- Computer Science Education (SIGCSE)
- Computer Uses in Education (SIGCUE)
- University and College Computing Services (SIGUCCS)

Association for Education Data Systems (AEDS)

American Educational Research Association/Special Interest Group on:
- Computer Applications in Instruction (AERA/SIGCAI)

AFIPS Education Committee

Association for Computers and the Humanities (ACH)

Association for Small Computer Users in Education (ASCUE)

American Society for Engineering Education/Computers in Education Division (ASEE/CoED)

Conference on Computers in Undergraduate Curricula (CCUC)

Educational Computing in Minority Institutions (ECMI)

Health Education Network (HEN)

International Council for Computers in Education (ICCE)

EDUNET/EDUCOM

IEEE Computer Society

Society for Computer Simulation

FOREWORD

This volume of proceedings of the Fifth National Educational Computing Conference (NECC/83), accurately reflects recent research and current trends in the field of computers and education. It embodies the critical thinking of a vast number of experts on topics that are both crucial to the whole society and especially relevant at the present time. The conference shows an excellent balance of reviewed papers, tutorials, panels, project presentations, and other sessions covering the broad spectrum of computers in education. The ideas expressed in these proceedings and in the sessions are a manifestation of the vitality of this field and provide attendees an opportunity to expand their expertise and increase their appreciation of computers in education. We believe that this conference will be beneficial to all participants and that these proceedings will serve as a valuable reference in the future.

The conference and these proceedings are the culmination of a great deal of effort by many individuals. Particular thanks are due to

- the National Educational Computing Conference Steering Committee for guidance and support, especially those who advised and encouraged the conference committee;
- all authors who submitted papers for review;
- the referees for their considerable efforts in reviewing the papers and for making the frequently difficult decisions of whether to accept or reject papers;
- the organizer of panel and tutorial sessions;
- A. J. (Joe) Turner (Clemson University), who so ably chaired the Program Committee and had the awesome task of coordinating the review of papers;
- Jean Rogers (University of Oregon), who with diligence and skill coordinated society sessions, tutorials and suggested sessions;
- William Ryan (Swarthmore College), who organized the project presentation sessions;
- William Dorn (University of Denver), who with discriminating sense (flair) organized the invited sessions;
- James Adams (Association for Computing Machinery), who with energy and imagination handled the publicity for the conference;
- Gerald Leach-Lewis (IEEE Computer Society), who worked creatively to expand the quantity and quality of the exhibits;
- Alan L. Roecks (San Antonio, ESC), whose superb evaluation of NECC/82 gave us excellent ideas for this years conference;
- Ralph Lee (University of Missouri, Rolla), who organized a splendid array of pre-conference workshops;
- Francis Edwards (Towson State University), who worked effectively on the broad range of local arrangement tasks;
- David Stonehill (University of Rochester), who organized the Birds of a Feather sessions;
- Robert Caret (Towson State University), who was always willing to assist and support the activities of the conference;
- Carol Edwards (Towson State University), who with good humor coordinated the processing of nearly 100,000 pieces of mail;
- Iva Thommen (Towson State University), who handled all secretarial tasks cheerfully and expeditiously;
- Donna Feldmann (Towson State University), who as the student assistant for NECC intuitively saw what needed to be done and efficiently did it;
- All attendees who made the efforts worthwhile;
- Della Bonnette (University of Southwestern Louisiana), who made this volume of proceedings possible, through her skills as an editor, her patience in dealing with the authors, and her ability to accomplish it all on schedule.

Doris K. Lidtke
General Chairperson, NECC/83
Towson State University
Baltimore, Maryland 21204

TABLE OF CONTENTS

THE ROLE OF LANGUAGE IN TEACHING PROGRAMMING

Stephen J. Garland
Dept. of Mathematics and Computer Science
Dartmouth College

ABSTRACT

When teaching students how to write, we must teach them how to write in a specific language such as English or French. When teaching them how to program, we must teach them how to program in a specific language such as Basic or Pascal. In both cases, a language is the vehicle, not the object, of instruction.

Teaching a language involves instruction in vocabulary, spelling, grammar, and punctuation. Teaching writing or programming, on the other hand, also involves instruction in logic, organization, expression, and style.

The reason language becomes an issue in teaching programming is simply that we have a choice. Students have learned their native tongue much before they learn to write, but generally they must learn a programming language when they learn to program.

A good programming language should enhance our ability to teach programming, not distract our attention from that task. It should:

be easy to learn, so that we can devote time to teaching programming and not just to teaching the language;

enable us to say what we want naturally and easily, so that we can write programs to fit problems, not to fit the language;

help us organize and convey our thoughts, so that we can understand and be understood;

be used in a uniform manner by many programmers, so that we and they can share our knowledge.

No programming language is perfect by these criteria. Basic is easy to learn, but most of its common dialects cause programmers to obscure, rather than illuminate, the structure of their programs. Pascal has fewer divergent dialects, and it allows us to express many constructs quite nicely; yet it can make other constructs extremely awkward.

The best teaching strategy is to turn this lack of perfection into an asset. Teaching the limitations of a language along with its virtues illustrates dramatically that programming transcends language.

APPROACHES TO REQUIRING MICROCOMPUTERS OF UNDERGRADUATES

Chaired by: Jane Caviness
University of Delaware

ABSTRACT: Approaches to Requiring Microcomputers of Undergraduates

The use of microcomputers is growing rapidly, while the age of the users and the cost of the microcomputers have been decreasing. Secondary schools are discovering that many of their students have acquired microcomputers and desire some general computing instruction. Colleges and universities are discovering that many entering students already have computing experience, most often with microcomputers, and they wish to continue using microcomputers. This presents a challenge to those involved in Computing Services, since they are accustomed to providing services through the use of timesharing on medium to large scale machines. How are they to deal with the change in the type of demand for computing services?

Answers to this challenge cover the spectrum from ingoring the problem totally, to turning it around and requiring undergraduates to have their own microcomputers. The panel members are all from institutions that have taken the latter approach. They will discuss many aspects of such an action: the decision to do so, the planning involved, the choice of hardware, the costs involved, the expected benefits, the difficulties of implementation, student reactions, and perceptions of first experiences. Discussion and questions from the floor are encouraged.

M. Peter Jurkat
Stevens Institute of Technology

Robb Russell
Drexel University

Wilson Dillaway
Rennselaer Polytechnic Institute

Doug Van Houweling
Carnegie-Mellon University

David Bray
Clarkson College

Accreditation in the Computing Sciences

John F. Dalphin, Moderator
Purdue University

ABSTRACT

A joint task force of the ACM and IEEE Computer Society is meeting regularly to discuss issues relating to accreditation or approval in the computing sciences. In addition to considering various mechanisms to implement the important qualitative review and certification, the Joint Committee is developing a preliminary set of Computer Science Program requirements.

Increasing requests are being made to the professional societies to provide guidance in computer science programs. While certain guidance and evaluation mechanisms exist, and agencies to administer them, these tend to be directed to specialized programs and the field is so broad that a wider view must be taken. It is estimated that as many as 500 programs not presently served by existing mechanisms and agencies would benefit from such guidance.

This panel will discuss some of the issues relating to implementation of accreditation or approval as well as quantitative criteria for computer science programs that provide competency in the profession. Audience participation and discussion will be encouraged.

PANELISTS:

Michael C. Mulder
Servio Logic Corporation

Tom Cain
University of Pittsburgh

George Davida
University of Wisconsin - Milwaukee

Gerald L. Engel
Christopher Newport College

Terry J. Frederick
University of Central Florida

Norman E. Gibbs
Arizona State University

Harvey Garner
University of Pennsylvania

SPONSORS: IEEE and ACM

NETWORKING FOR MICROCOMPUTER MANAGEMENT

Kenneth Forman
Carl Steinhoff

Community School District 27
New York University

A network of several microcomputers connected to a common hard disk storage system provides several administrative functions for effective management.

Community School District 27 is one of the 32 public school districts within the City of New York with approximately 27,000 pupils in grades K-9 and 1,400 employees. Early in 1980, we began to investigate the feasibility of using computers for evaluative and management purposes. This investigation was a collaborative effort with our evaluation consultant, New York University, under the leadership of Dr. Carl Steinhoff. We determined that microcomputers would most effectively manage all the various applications we desired to implement for fiscal, information and evaluative reporting.

Several microcomputer systems were investigated including: Apple, Atari, Bell and Howell, Commodore/Pet, Radio Shack, etc. Upon reviewing the literature, we came upon the successful experiences of MECC (Minnesota Educational Computing Consortium) with Apple microcomputers. The Apple microcomputer offered numerous applications in the areas of business, financial reporting, personnel and information management, as well as its ability to interface with larger systems. Therefore, we decided not to "reinvent the wheel", but to improve on the applicability of Apple microcomputer systems. Our plan involved creating a microcomputer network for administrative and fiscal reporting.

With the support of our Community School Board and District Superintendent, Marvin R. Aaron, we were ready to implement our microcomputer network design (see Appendix).

Networking refers to connecting several microcomputers together through a common transmission line and central source so as to allow the sharing of information and peripheral devices (mass storage, printer, modem).

Devices connected to a network have been termed "nodes" by network users. Currently, nodes must be intelligent. They must not be individual microcomputers or intelligent peripherals (printer/modem)[1].

Current research defines three basic types of networks: Star, Daisy Chain and Drop Line (Bus)[1,2]. A microcomputer network constructed in Star configuration, consists of a central intelligent microcomputer, termed the host, with other devices connected in a radial or starlike pattern. All devices are directly connected to the host. If one device becomes inoperative, other devices still function. A Daisy Chain configurated network consists of a single host with all other devices wired in series to the host. If one device becomes inoperative, all devices past the inoperative device cease to function. The third network type, a Drop Line or Bus configuration, consists of a host with a single cable. All devices are connected in parallel to the main cable via junction boxes so that failure of any one device will leave the remainder of the system operative.

Our investigation of types of networks available for use with microcomputers led us[3] to select the "Omninet" of Corvus Systems, Inc. Omninet is constructed essentially in a Drop Line configuration, with each intelligent device connected to a hard disk storage system via an easily installed piece of hardware termed a "transporter". The advantage of this type of network is obvious, if a problem arises in one intelligent device, all other devices remain functioning. All intelligent devices share the resources of hard disk system. In addition, a primary intelligent device or host can control access of other users through a user defined security system, giving users (up to 64) different levels of access to information; read only, read and write and manager level access for security purposes.

Therefore, our initial microcomputer network design consisted of the following equipment: 5 Apple II Plus (48k) Microcomputers with Language Cards, 5 Disk Drives with Controllers, 5 Zenith Data Monitors, 1 Qume 5 Printer, 1 Corvus 10 Megabyte Hard Disk System, 1 Corvus Disk Server, 8 Corvus Transporters, Panasonic Video Cassette Recorder (NV 8200) and D.C. Hayes Micromodem (see Appendix for approximate costs).

Our financial commitment toward developing

a microcomputer management network for district use was further supported through creating a district position of "Computer Specialist", that is, a person to provide support in implementing this network design.

To allow for rapid implementation of our management network, we chose to use commercially developed software packages developed for hard disk systems rather than have our Computer Specialist develop customized management software (which would have delayed implementation over several months). These packages include DB Master (a data base management package), Word Handler (a word processing program) and Visicalc (a numerical data manipulation program). DB Master was selected for our data base management for several reasons, some of which include: automatic data compaction upon storage, sophisticated report generation and password protection system.

Let's take another step back and discuss methods of data base management. Data base management can be viewed as a pyramid structure. In a top/bottom structure, all information is collected by a central authority for creation of a data base. Subsequently, information is reported to collection sites for verification. In a bottom/top structured data base, each site manages its own data base. We used the floppy disk version of DB Master, which is, of course, compatible with the district hard disk version used for data base management. In a bottom/top configuration, each site shares its data base with the district producing a more accurate data base. Each site has a vested interest in management of its own information. District supports each site and a cooperative working partnership has developed.

Each site manages its data base using the following equipment: Apple II Plus (48K) Microcomputer with dual disk drives, Zenith Data Monitor, Epson MX 80 Printer and DB Master (floppy version). Recently, we have provided selected sites with DC Hayes Micromodems for eventual telephone hookups. Furthermore, for more efficient use of information management, sites are also adopting word processing using Word Handler (floppy version) which is interactive with our data base management program.

Our microcomputer management network functions to support the instructional process through various management applications, which include:

- word processing
- information storage and retrieval
- inventory
- mailing lists/labels
- vendor reports
- personnel records
- student records
- ad hoc reporting from larger data files

Individual microcomputers within the network can function independently, and with a small hardware attachment, can also function as a remote intelligent terminal with the Central Board of Education's IBM mainframe computer.

Recently, we have begun to experiment with an optical scanning device, the Scantron 2700, for mass entry of information into our data base. Eventually this device will minimize data entry time permitting immediate use of a data base at each participating site as well as facilitating creation of a master district data base.

Additional software applications for our network will be forthcoming; for example, we are investigating general accounting software (accounts payable, receivable, general ledger) which will be interactive with other programs. As an additional support to participating sites, we are investigating student attendance and scheduling programs that would be interactive with our student data base.

Lastly, one must consider confidentiality of information, security precautions and levels of access to information within a network. We have structured Omninet with four levels of security protection to authorize only approved users access to the network. First, Omninet only recognizes authorized users by an individual "name" assigned to each user; illegal users are denied access to the network by Omninet. Once the network recognizes the user's "name", the user must then enter an identification code for further access to the network. Then, authorized users must place the DB Master Management Disk into the disk drive to access the data base management program stored within the hard disk system. Finally, the user must enter another password to gain access to data files. Once within the network, Omninet is structured to permit differentiated levels of access, that is, read, read and write, and manager level access to information.

If networking will best suit your management needs, then consider the following questions[4]:

1. What is the greatest distance from one end of the network to the other?
2. What is the maximum number of microcomputers to be networked?
3. How much mass storage is required?
4. Which network offers capability for expanding or upgrading?
5. Are there current users that I can speak with?
6. What kind of service and support are available?
7. Is the manufacturer reliable?
8. Is the product supported by on-going development?

9. What is the cost of installation?

10. Can microcomputers of different manufacturers and/or peripherals be connected within the network?

REFERENCES

1. Minnesota Educational Computing Consortium (MECC), "Spotlight on Local Networking with the Apple II Computer", September 1982.

2. Charp, Sylvia, "Trends - Time Sharing, Microcomputers - Networking", T.H.E. Journal, November, 1981.

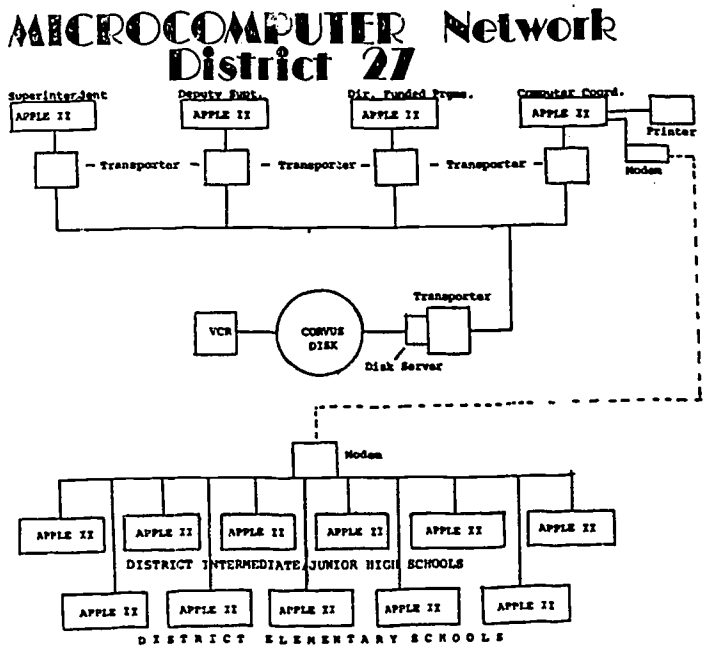
3. Corvus Systems, "Winchester Disk Systems for the Apple II and Apple II Personal Computers", Corvus Systems, San Jose, California, 1982.

4. Connell, Cassie, "Networking" What are the Alternative Systems", T.H.E. Journal, September 1982.

APPENDIX

Omninet Hardware Costs

5 Apple II Plus (48K) Microcomputers 1146 X 5	$5730.
5 Apple Disk Drives/ Controllers 483 X 5	2415.
5 Apple Language Cards 146 X 5	730.
5 Zenith Data Monitors 135 X 5	675.
1 Qume 5 Printer (with cable/interface)	2245.
1 Corvus 10 Megabyte Hard Disk System	3495.
1 Corvus Disk Server	990.
8 Transporters (1895/4 units) (1895 X 2)	3790.
1 Panasonic V.C.R. (NV - 8200)	1000.
Wiring/Installation	1000.
TOTAL	$22070.



SPREAD SHEET SIMULATION MODELING (SSSM) FOR TRAINING AND INSTRUCTION IN RESOURCE ALLOCATION

Ronald A. Lindahl, Assistant Professor
Brent E. Wholeben, Associate Professor

Department of Educational Administration and Supervision
The University of Texas at El Paso (El Paso, Texas 79968)

With the emergence of microcomputer technology over the past decade, managers in all fields have found powerful new tools at their disposal to assist them in decision-making. One of the most utilitarian and universally accepted of these new resources has been the electronic spread sheet. However, only recently have the programs responsible for the initial preparation and continued development of such managers have only recently begun to appreciate and explore the value of incorporating spread sheet simulation modeling (SSSM) into their instructional programs. The very same features of flexibility, adaptability, and facility of operation which mark the usefulness of electronic spread sheets for managers are equally relevant to their value as an instructional device.

Spread Sheet Simulation Modeling

Electronic spread sheets have tremendously expanded the manager's capabilities beyond traditional paper, pencil, and calculator worksheet methodology. Prior to the advent of microcomputer technology and the development of appropriate software, spread sheets were common managerial tools for preparing and managing budgets, making financial projections, forecasting sales and profits, and controlling inventories, etc. These spread sheets were basically a grid of columns and rows, organized in such a manner as to allow the manager to view both horizontal and vertical interrelationships between cell components of the grid.

More sophisticated applications called for the master grid to be divided into various semi-independent sub-matrices. Under such an arrangement, the manager could perform all calculations in one sub-matrix, record a table of figures (e.g., a tax table, amortization table, or depreciation index) in another sub-matrix and maintain a third sub-matrix as the primary exhibit of transactions.

Consequently, today's manager has the ability to be able to make virtually limitless changes in the worksheet in order to witness instantaneously the sensitivity of related components to these modifications. Where entering a new value onto a traditional "paper and pencil" worksheet might require the recalculation, erasure, and re-entry of numerous interrelated figures, the electronic spread sheet requires only a re-entry of the initial figures; all subsequent impacts would be reflected automatically.

This apparently simple technological advance has dramatically altered the manager's perspectives for simulation modeling and sensitivity testing. No longer faced with hours of tedious recalculations and revisions for each desired change, and the inherent potential for clerical error, the manager now has access to a spread sheet which facilitates and encourages manipulation and creative experimentation, without sacrificing its integrity as a tool for data registration and analysis.

SSSM as an Instructional Resource

However significant a tool SSSM may be for today's managers, it proves to be of even greater worth as an instructional aid in the preparation and development of these managers.

One of the key features of the electronic spread sheet is its relative simplicity, a characteristic also of prime importance in considering SSSM for instructional purposes. It is not surprising that a significant proportion of the existing managerial labor force has not yet had the opportunity to become "computer literate." However, it should also be noted that a significant proportion of the professors, instructors, trainers, and consultants engaged in the preparation and development of managers is similarly unacquainted with this new technology. Consequently, for SSSM to be incorporated successfully into instructional programs, both instructors and students must be able to grasp its basic technological aspects with a minimal investment of time and effort. While most major producers of software offer their own version of the electronic spread sheet, many share common features; and most are structured and documented well enough that novices can readily master the rudiments of their operation. Obviously, some of the more sophisticated features may require in-depth study and

hands-on experimentation; however, their use would be optional and need not be a deterrent to the introduction of SSSM as an instructional approach.

Another feature of electronic spread sheets which has facilitated their acceptance among managers is that of versatility. Managers have exhibited considerable creativity in developing new ways to use electronic spread sheets in solving problems and making decisions. Rather than learning the intricacies of several different, dedicated software programs to address specific applications, managers have found the spread sheet to be versatile and adaptable to a wide variety of situations. The savings, in terms of time and energy required to master the rudiments of the program and make it operational, have been significant.

Obviously, this feature is also of prime importance for instructional applications. Having mastered the rudiments of the electronic spread sheet, the instructor can successfully employ it for a variety of simulations without having to invest further time on the operational aspects. At the same time, students who have familiarized themselves with the basics of the spread sheet can better concentrate on the content aspects of all subsequent simulations. In view of the time constraints and need for instructional efficiency which characterize most managerial science programs, this is a highly significant feature.

The Value of Simulation Modeling

In his book, Computer Modeling and Simulation, Francis Martin defines simulation modeling as "a logical-mathematical representation of a concept, system, or operation programmed for solution on a high-speed computer." In Computer Simulation Applications, Julian Reitman describes simulation as "a practical, application oriented procedure" in which "one must construct an abstraction of the problem, transfer the problem to a foreign device, the computer, and then obtain indications pertaining only to the representation of the system." Both authors then dedicate considerable discussion to the problem of identifying those conditions under which simulation modeling can effectively be employed.

Simulation modeling enables the manager, professor, or student to: (1) examine a complex situation; (2) state the problem in a simplified, understandable form; (3) define the rules and relationships which govern the situation; (4) and experiment with operating and manipulating the system. As Reitman points out, this affords several significant benefits, including: "aid in problem definition; help in relating numerous factors with their influences on the design; insight into the sensitivity of the design to wide ranges of parameters; support for selecting the final design from among the alternatives uncovered; and guidance in predicting system performance."

Basic SSSM: Cohort Survival Analysis

As discussed earlier, the spread sheet may be configured as a single, master matrix or, for more complex situations, as a composite of several interrelated sub-matrices. For less complex models, the single matrix format facilitates an understanding of the whole, while still allowing the student to perceive and follow the interactions.

One example of the successful utilization of a single matrix in the instruction of resource allocation is in the teaching of cohort survival analysis. Cohort survival analysis is a forecasting technique in which the survival, attrition, and new membership rates of certain specific population groups are examined over successive time periods. By extrapolating this data, and assuming a continuity of past trends, projections can be made regarding the future population (membership) of the group.

The cohort analysis technique has been effectively employed in the forecasting of school enrollments in those districts which have achieved some consistency or stability of population growth or decline. The matrix consists of both historical and extrapolated data, with the columns being defined as the years under consideration and the rows being configured as the grade levels being examined. The individual cells contain the actual class membership figures for past years and formulae to predict a weighted average for future years. The computer then automatically calculates the projected enrollment for future years, either by linear extrapolation of past trends or by systematically weighting the most recent trends more heavily than earlier performance of the system. Figures 1 through 3 illustrate the three major phases of this SSSM process: (1) entering background data; (2) derivation of survival ratios; (3) and projection of future enrollments.

At the same time, it is extremely simple for students to construct a matrix which would allow them to change this formula, enter in their own survival ratio predictions, and test the sensitivity of the projected enrollment to such variations. Figure 4 illustrates a simplified version of such a learner-manipulated matrix.

Although the above example serves to illustrate the utility of the SSSM in its most simple form, it by no means exhausts the capability of this tool in helping students understand cohort survival analysis. For example, it is a relatively simple matter for the professor to set up the matrix as an interrelated series of submatrices, which the student can study independently to better understand the various components of the cohort survival analysis process. One such sub-matrix which has not yet been demonstrated, but which would be of great value in training students to utilize this resource allocation tool, would be that of multi-year forecasts. Figure 5 presents such an extended projection, based on the assumption that the trends upon which the single-

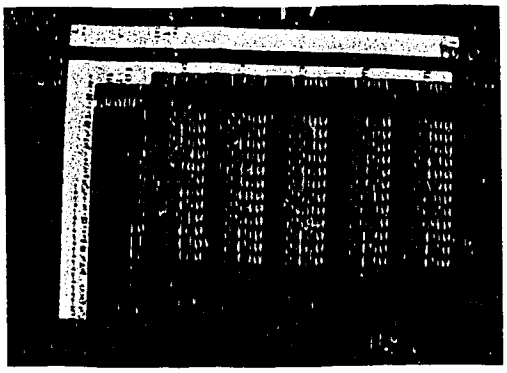


Figure 1.
Background Data

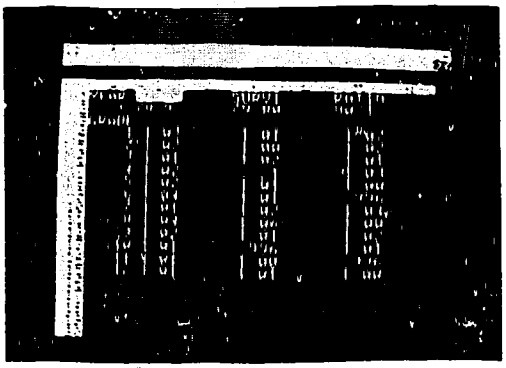


Figure 2.
Survival Ratios

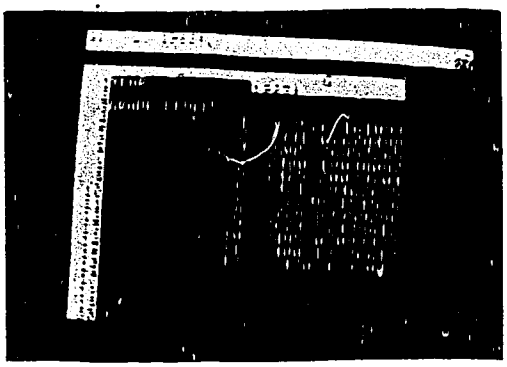


Figure 3.
Enrollment Projections

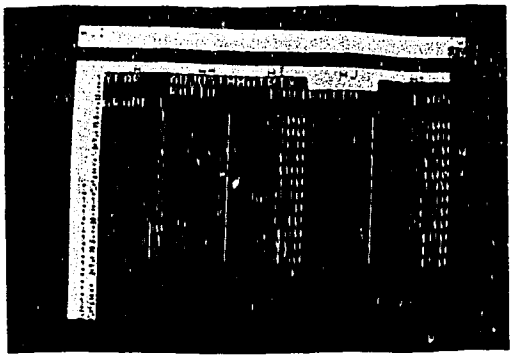


Figure 4.
Learner-Manipulated Survival Ratios

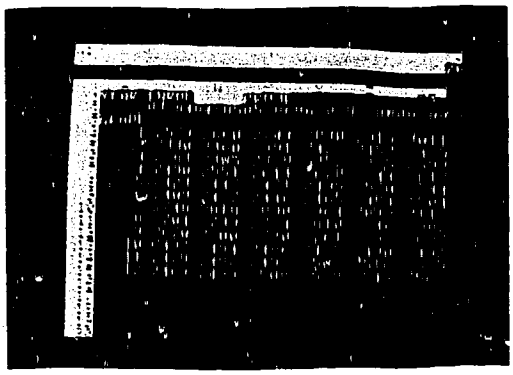


Figure 5.
Multi-year Enrollment Projections

year projection were based would continue in a linear pattern throughout the total period being considered.

Extended SSSM: State Allocation Formula

In light of the flexibility and capacity of such multiple sub-matrix models, there are a myriad of instructional uses which can be served by SSSM, especially in the area of resource allocation. For example, students at The University of Texas at El Paso have responded very positively to the use of this tool in investigating the predicted effects of changes in State legislation on the allocations of operating funds to local school districts.

In this application, the electronic spread sheet is configured as an inter-related composite of sub-matrices. The principle sub-matrix presents a summary of the major components of the State allocation formula, including the:

- flat grant stipend for each unit of average daily attendance;
- State contribution to personnel costs, based on the maximum allowed number of weighted personnel units and the State minimum foundation salary scale;
- special program supplemental funds, for such areas as special education and vocational education;
- transportation allotments, again differentiating between regular and special program costs;
- State compensatory funds, providing program enrichment for children of low-income families;
- cost-adjustment allocations for small and/or sparsely populated districts; and,
- State minimum foundation program funds for district operation and maintenance expenses.

By limiting this principal matrix to a flow-through listing of the total allocations available from each of these components, the student is better able to view the formula from the systems perspective. Concomitantly, a separate sub-matrix would be extablished for each component, so that the student might clearly understand how the allocation is generated. This sub-matrix system also facilitates the sensitivity testing of minor changes in any given component which might be proposed by the Legislature. For example, one sub-matrix would consist of data on average daily attendance at the various grade levels identified in the State formula.

Calculations would be shown in this matrix to illustrate how these figures are converted into weighted personnel units. On those microcomputer systems which would afford sufficient memory to accommodate a matrix of sufficient size, e.g. greater than 64K, the State minimum salary matrix can be entered as a separate sub-matrix. This, in turn, can then be referenced directly from the sub-matrix created to reflect the staffing pattern selected by the district. Another sub-matrix would be designed to record the participation of students in special programs; the calculations employed in this matrix would then feed into the principal matrix to indicate funding for classroom units, additional weighted personnel units, etc..

Such a SSSM structure allows the students to manipulate any of the variables of the formula alone or concomitantly and to test the sensitivity of the overall allocation to such modifications. For examples, students can compare the several proposals which will be considered for adoption at the next Legislative session and determine the effects of each on local district allocations. At the same time, they can examine the financial implications of varying the district's staffing pattern, test the sensitivity of the allocations to various enrollment projections, or make inter-district comparisons, all with great ease.

Figures 6 through 8 illustrate typical screen formats from a simplified version of this application.

SSSM: Instructional Transition to Application

One of the most significant benefits of employing SSSM for instructional purposes is the high degree of transferability of knowledge, allowing the learner to make the transition from the academic environment to managerial decision-making reality.

The relatively low cost of microcomputer hardware, coupled with the almost universal availability of electronic spread sheets for all major configurations of this hardware, suggests that it would not be unreasonable to expect that a high proportion of today's managers would have access to this technology within their normal work environments. Previous "generations" of learners were faced with tremendous difficulties in attempting to transfer their experiences on the mainframe equipment utilized in the instructional environment to often radically different configurations of equipment within their professional work environments. With the accessability of microcomputers in virtually all professional settings, and with the high degree of transferability of knowledge between the various electronic spread sheet/microcomputer hardware configurations, today's students' time investment in learning the basics of SSSM may well be translated into increased productivity upon entry into or return to the work situation.

As such, the manager can personally configure electronic spread sheets to address specific real decisions. At this point, the learning tool is transformed into a decision-making aid, allowing the manager to combine professional expertise with the data analysis capabilities of the SSSM to

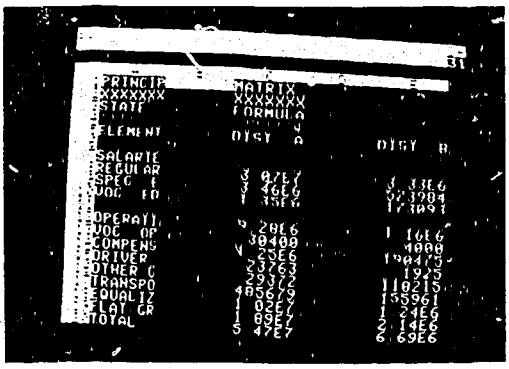


Figure 6.
Principal Calculating Matrix

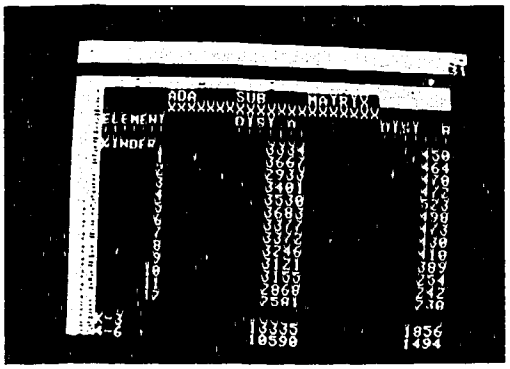


Figure 7.
Average Daily Attendance Matrix

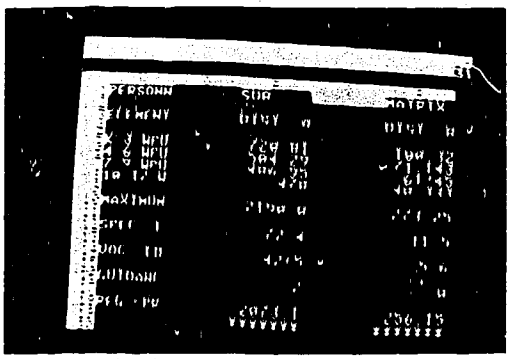


Figure 8.
Weighted Personnel Unit Matrix

simulate the situation in question under a variety of assumptions and hypotheses. The cycle, in turn, becomes complete when the learning and discoveries gained through these practical applications of SSSM are brought back into the instructional setting to enhance the education and re-cycling of other managers.

Summary

In conclusion, educators responsible for the training and instruction of topics related to the area of resource allocation will find Spread Sheet Simulation Modeling to be a versatile, effective instructional technology. In addition to its versatility and applicability to a wide range of managerial decisions, SSSM offers the instructor the desirable characteristics of being relatively simple to learn, even for the non-computer-literate, and of promoting interaction between the learner and the technology.

At the same time, students who become familiar with this technology in the classroom setting will actually reap a double benefit; not only will they have the opportunity to better understand the content area being presented, but they can also become conversant with a powerful decision-making tool which can be of invaluable assistance to them throughout their managerial careers.

Bibliography and Related Readings

1. Francis F. Martin, Computer Modeling and Simulation. (New York: John Wiley & Sons, Inc., 1968).

2. Julian Reitman, Computer Simulation Applications. (New York: John Wiley & Sons, Inc., 1971).

3. Brent E. Wholeben, "Operational Network Displays" in Communication Strategies in Evaluation (Nick L. Smith, Ed., Beverly Hills, California: SAGE Publications, 1982).

4. Brent E. Wholeben, The Design, Implementation and Evaluation of Mathematical Modeling Procedures for Decisioning Among Educational Alternatives. (Lanham, Maryland: University Press of America, Inc., 1980).

DEVELOPMENT AND VALIDATION OF A COMPUTERIZED ADAPTIVE SCREENING TEST (CAST) FOR USE IN ARMY RECRUITING

Herbert George Baker, PhD, Bernard A. Rafacz, and William A. Sands

Navy Personnel Research and Development Center
San Diego, CA 92152
619-225-2408

Abstract

NAVPERSRANDCEN designed and developed a Computerized Adaptive Screening Test (CAST) that predicts an applicant's score on a selection test. CAST is capable of operating on a stand-alone microcomputer in Army recruiting stations, and correlates at .866 with the criterion. Further data collection and validation efforts are currently in progress. This development suggests that a shorter, adaptive test is feasible for use by organizations employing an aptitude test in the selection processes.

Introduction

The Navy Personnel Research and Development Center (NAVPERSRANDCEN), through an agreement with the U. S. Army Research Institute for the Behavioral and Social Sciences, is conducting research and development in support of the Joint Optical Information Network (JOIN) System being implemented nationwide at all levels by the U. S. Army Recruiting Command (USAREC). Army enlisted applicants will directly interact with this system in the course of recruiting and accessioning. The objective of this effort was to design and develop the Computerized Adaptive Screening Test (CAST) that could: (1) operate on a stand-alone microcomputer system in recruiting stations as part of the JOIN system, (2) reduce recruiters' administrative burden, and (3) predict Army applicants' scores on the ASVAB more efficiently than the paper-and-pencil Enlistment Screening Test (EST) used at present.

Armed services recruiting faces serious challenges in the future, due to a shrinking pool of military eligibles.[1] The All-Volunteer Force (AVF) concept has led to vastly increased expenditures in recruiting.[2] A fierce and costly competition for available personnel among colleges and the several armed forces is probable, a competition that will increase the difficulty of recruiting.[3] The best available candidates for enlistment must be located, enlisted, and optimally assigned; neither fiscal nor personnel resources can be wasted; and those mundane tasks of the recruiter, which detract from the primary mission of locating propects and "selling the service," must be reduced.

The screening of applicants for enlistment, which takes place near the end of the recruiting process, includes as a major element the Armed Services Vocational Aptitude Battery (ASVAB), a 10-part test given to all armed services applicants. To be eligible for enlistment, an applicant must achieve a minimum qualifying score on the Armed Forces Qualifying Test (AFQT), which is a linear composite of the scores obtained on four ASVAB subtests. The ASVAB is given either through the Department of Defense High School Testing Program, or to service applicants at a Military Entrance Processing Station (MEPS) or Mobile Examining Test (MET) site.

Applicants who have no previous qualifying AFQT scores must be sent to the MEPS/MET for ASVAB testing. For those who must be transported from a recruiting station to a MEPS/MET site, costs are entailed for transportation, and in many cases for meals and lodging. Costs are also incurred for personnel time at the recruiting station. If applicants are sent to the MEPS/MET site and subsequently fail the ASVAB, there is a significant waste of money. Conversely, if applicants who would have passed are erroneously denied ASVAB testing, their talents are lost to the service, recruiting quotas are not met, and additional costs accrue to the applicant.

Currently, all armed services use the Enlistment Screening Test (EST) at recruiting stations to predict applicant AFQT scores. This test is composed of 3 subtests of 15 items each, with a total

---

*The views expressed herein are those of the authors and do not necessarily reflect those of the U. S. Navy.

time limit of 45 minutes. It was developed by the Air Force Human Resources Laboratory in 1976,[4] and revised in 1981.[5]

Pure measurement error does not seem to be a problem; it appears that the newer forms of the test (81a and 81b) measure quite well throughout the score range where most selection and classification decisions are made, and it correlates at .83 with the AFQT.[5] The EST, however, does share in the problems of adverse psychological effects. The EST is a conventionally administered paper-and-pencil test, which has been shown to increase guessing, frustration, and boredom in subjects.[6,7] In its customary delivery mode in the recruiting station, the EST actually combines the worst features of an individually administered test (i.e., it is heavily dependent upon variables associated with the examiner or the examiner-examinee relationship) and the group test (i.e., item arrangement, item set, answer sheet effects, and imprecision).[8] It is scored on a pass/fail basis. The EST, being a timed test, also exerts differential pressure on the examinee (i.e., results are based partly upon the reaction of the individual to time constraints.[7]

Major concern focuses on administrative error and clerical burden. The EST requires approximately 45 minutes to administer, as well as time to score and interpret results (by a recruiter already investing many hours in the potential enlistee). To this must be added the time required to manage the test supplies. At present, besides storing, filing, retrieving, and ordering replacements, the recruiter is required to take frequent inventory, make numerous checks and corrections for unauthorized markings in the test booklets, and safeguard used answer sheets. The EST is thus highly labor intensive, consuming the time of a senior noncommissioned officer in quasi-clerical tasks.

Only two forms of the EST are in use, a situation that offers the failing subject hope of eventually passing the test by repeated testing and item memorization. Furthermore, the recurrent problem of "malicious error" appears, involving recruiter malpractice. Recruiter influence through pretest coaching is a simple way to manipulate results. Initial and replacement costs and short materials life associated with paper-and-pencil tests; poor impression created by dilapidated materials; and security, custody, and control difficulties. In short, there is a critical need for both a more efficient predictive instrument and a screening method less burdensome in its administration, to reduce the recruiter's task load.

Computerized adaptive testing combines recent developments in latent trait theory with the ever-increasing power and efficiency of computers. The result is a return to individualized testing without the loss of administration efficiencies gained through group testing, combining advances in computer technology with those in psychometrics.

Only a limited subset of test items is needed to establish an aptitude estimate. In adaptive testing, each subject receives only those questions appropriate to his or her ability level. The result is an individualized test, "tailored" or "adapted," and actually constructed for each examinee. While there are several "branching" strategies available, in tailored testing the selection of each question is based on the subject's response to the previous question. Typically, a correct response is followed by a more difficult item; an incorrect response is followed by an item of less difficulty. Therefore, the difficulty level of an adaptive test is dynamically tailored to the ability of each subject. And, because each subject receives a tailored subset of items, the chances of test compromise through copying, memorization, or coaching are considerably lessened.

With adaptive testing, shorter tests may be used without loss of reliability or validity.[9,7] Adaptive tests are untimed, reducing pressure on the subject, without hindrance to the proctor.[10] Adaptive testing reduces boredom and frustration,[7] guessing,[9] real or perceived proctor-subject bias,[11] and culturally specific racial bias.[12] Adaptive testing is more motivating, thereby eliciting "best results" from subjects[13] and more accurately reflecting subject competence.[12]

Computerized testing has its own merits. Interactive dialogue provides immediate results without manual scoring. Immediate knowledge of results has been shown to be a motivator to better performance;[13,14,10] Computerized testing lessens test bias through item selection and increases test fairness by the nature of the test itself and the test's administration modality.[15] The computer makes it possible to eliminate printed test materials and their associated logistical, security, and administrative problems, while facilitating item replacement, whole test construction, and the capturing of data

for validation purposes.

Stated simply, the computer can administer a test item, accept a response, score that response, and keep a record of the subject's response history. After each question, the computer can use the response information to update the ability estimate and then use the new estimate of ability to select the next item. With each successive response, the ability estimate gains reliability. The process can continue until some stopping rule is satisfied (e.g., a fixed number of questions administered, or a prespecified level of reliability). Computerized adaptive testing can effectively shorten testing time without effectiveness loss as well as eliminate scoring and recording errors due to clerical error.[11]

CAST Design and Development

Designing and developing CAST required parallel work in psychometrics and computer programming. All design work was accomplished on an Applied Computer Systems (ACS) microcomputer with a Perkin-Elmer Data Systems 1200 video display terminal (VDT). The effort proceeded in the following steps:

Test Construction

CAST was envisioned as incorporating three subtests that would correspond to three of the four ASVAB subtests used to calculate the AFQT composite score: Word Knowledge (WK), Arithmetic Reasoning (AR), and Paragraph Comprehension (PC). These subtests were determined to be the best predictors of AFQT score. Item banks for each subtest were assembled under a contract with the University of Minnesota. These item banks included 78 WK items, 247 AR items, and 25 PC items, together with the estimates of three parameters (discrimination, difficulty, and guessing) for each item. A Bayesian ability estimation procedure described by Jensema[16] was chosen for scoring and determining the selection and presentation sequence of test items.

Computer Software

CAST computer programs were written to provide interactive, user-friendly software that presumed no previous computer experience on the part of either recruiter or applicant. In addition, VDT screen text displays were written to conform to readability (reading grade level) standards.

Test Administration

In each CAST subtest, a provisional ability estimate is made, a test item appropriate to that ability level is presented, and then the ability estimate is updated based on the response to the test item. The computer program for this iterative ability estimation process starts by associating each examinee with an ability level of the information matrix. Selection of a test item starts from the top (highest information value within a level) and searches for the first item not yet presented. The item that results from this search is then presented. Based on the examinee's correct or incorrect response, the ability estimate is updated. This associates the examinee with a new ability level of the information matrix, (potentially) from which a new test item is selected for presentation. This process continues until the limit established by the stopping rule is reached.

To facilitate user acceptance, the time between item response and presentation of the next item was established as less than or equal to three seconds. Software documentation for presentation of CAST on the ACS microcomputer system was completed. Subsequently, the program was converted to operate on the JOIN developmental system, an Apple II-Plus microcomputer, with two disk drives, and a VDT.

Test Validation

Concurrent NAVPERSRANDCEN research to assess the relationship between paper-and-pencil ASVAB tests and an experimental battery of three computerized adaptive subtests (WK, AR, and PC) provided the opportunity to conduct a pilot test of CAST. The three CAST subtests were administered during a 90-day period late in 1981 to 356 male Marine Corps recruits at the Marine Corps Recruit Depot, San Diego. Each recruit had taken the ASVAB before enlistment and had been retested on a parallel form of ASVAB during recruit processing. After eliminating subjects with missing scores on any test and those who had been administered obsolete forms of ASVAB, the remaining sample was 270.[17]

In the CAST pilot test, each subtest was administered with a fixed length: 15 items each for WK and AR, and 10 items for PC. All examinees began with the same item, which was of medium difficulty. While the paper-and pencil ASVAB is a timed test, the adaptive tests were conducted without time limit. Preliminary introduction to the testing

situation was delivered orally by the proctor, while all other instructions, including use of the terminal and procedures for answer entry and answer changes were delivered on the terminal screen. Practice preceded each subtest; the successful response to these items was a condition of beginning the subtest. The test was administered by computer, on four terminals in a specially designated testing room. The terminals were on-line with the Hewlett-Packard 21 MX computer at the University of Minnesota, through a data communications line.

There were several differences between CAST as it was designed to be given and the subtest administration during the pilot test described herein. A true backspace key was not available on the terminal, requiring the recruit subjects to use the "Rubout" key" to make corrections. In the PC subtest, the stimulus paragraphs did not remain on the screen while the response alternatives were displayed. The pilot test was administered on terminals communicating with a host computer, while CAST is designed for use on a stand-alone microcomputer with attached disk drives. Finally, because of restriction in range, the Marine Corps recruit test subjects are not representative of an unselected applicant population.

After test scores were collected, the relationship between CAST subtests AR, WK, and PC and their paper-and-pencil counterparts was evaluated through correlational analysis. The ability of the CAST subtests to predict AFQT composite (AR, WK, PC, + 1/2 Numerical Operations) scores was assessed using multiple regression analysis.

Results

Each CAST subtest correlated as well with its ASVAB counterpart as did the parallel form ASVAB retest score. Multiple regression analysis to evaluate the relationship of the CAST subtests with AFQT score indicated a multiple correlation of .866

Notwithstanding the differences mentioned previously between CAST administration as originally designed and as carried out in the pilot test, the results of the pilot test are significant. It was clearly demonstrated that military recruits, and by implication, military applicants, could be tested by computer terminal with minimal intervention by a proctor. The CAST subtests measured the same abilities as the corresponding ASVAB subtests, but with about half the number of questions.[17]

Work in Progress

Data analyses are now underway to evaluate the validity of CAST with its intended population. The CAST was administered to 364 Army applicants at the Los Angeles MEPS between 29 November 1982 and 7 January 1983, using Apple II Plus microcomputers operating under CPM, with two disk drives and a VDT. Results are pending completion of analyses.

Discussion

A complete microcomputer-based CAST demonstration system is now in operation, with complete documentation for all software. The test item bank for AR has been reduced in size to 225 items. The WK and PC item banks remain at 78 and 25 items respectively. User-friendly, interactive software provides full screen display, clearing the screen after each display. Response time is three seconds or less. An easily used backspace key, feedback to the user after each answer, and an error-trapping capability that ensures recruiter assistance after repeated procedural errors by the applicant have been added to the system.

Remaining research and development in adaptive testing involves evaluating the interactive screen dialogues for readability and user friendliness with actual applicants or recruits, and field testing the CAST system in a recruiting station. Capitalizing on the results of the pilot testing of the subtests, it appears entirely feasible to eliminate the PC subtest from CAST, without a significant decrement in the correlation between the CAST linear composite and AFQT (see Table 1).[17] Examination of the data suggests that the length for the AR and WK subtests could be set at 7 and 15 items respectively, without appreciable loss in predictive validity. Were both procedures to be implemented, the correlation between AFQT and CAST would drop only to .865 (from .866), still comparing very favorably to that of the current EST (.83). The result would be a CAST of only two subtests, requiring an average of 16 minutes for complete administration, scoring, and interpretation, as opposed to 45 minutes for the EST administration alone, a savings of approximately 65%.

Conclusions

CAST can be regarded as successfully developed, requiring only minor refinements in both psychometrics and programming. It is superior to the EST in terms of administration and management and about equal in predictive power. CAST eliminates the need for traditional test materials, thereby saving storage

space, replacement costs, and recruiter time formerly used for administering the tests and controlling and maintaining test materials. With CAST, test loss, theft, and compromise would be all but eliminated. Security would be maintained by a built-in user password or identification. Rather than serving as a test proctor-scorer, the recruiter simply would manage a computer-subject dialogue, with a self-scoring test for which results are immediately available and may be automatically stored for later use.

Cost effectiveness would result from eliminating traditional test materials and from reducing recruiter time spent in testing. In short, CAST will decrease negative psychological effects, decrease administrative error, and increase test security. CAST is important for its present economizing service, but important, too, for its enabling functions. Implementing CAST will allow the Army to be highly responsive to advances in psychometrics and managerial science, as well as enable it to implement further applications rapidly when they are needed. Computerized ability testing systems are predicted to find their optimal use in organizations serving populations of wide-ranging ability,[18] and with CAST, the Army will be in the forefront of Computerized Adaptive Testing (CAT) implementation.

In today's recruiting climate, where increased screening capabilities assume ever greater importance, the Army will have the technological base upon which to mount other screening instruments for both selection and classification. These might include: predictors of tenure and effectiveness;[19,20,21] assessment of expectations, intentions, job perceptions, and attitudes;[22] and screening for specific placement. Screening could be significantly improved by expanding the array of measures to include special abilities and, even biodata,[23] since the administration and motivational problems associated with lengthy testing and examinee fatigue would be reduced by automation.

Tangentially, future test development costs and intrusion on operating systems will be reduced because experimental test items can be introduced within CAST, in a manner that is transparent to the field user. This will facilitate the development and evaluation of new items. Test administration will be standardized, fairer to all applicants, and far more efficient in scoring and recording methods.

Importantly, CAST demonstrates that any organization using an aptitude test battery in a selection environment may profit from development of a shorter, adaptive test to predict success on the battery. There is nothing inherently military about the ASVAB or the CAST. This research may therefore be generalized in large measure to analogous situations.

References

1. Congressional Budget Office. Costs of manning the active duty military (Staff working paper). Washington, DC: Author, May 1980.

2. Office of Naval Research, Psychological Sciences Division. 1979 programs (450-11). Arlington, VA: Author, November 1979.

3. Joint Chiefs of Staff. United States military posture for FY 83. Washington, DC: Author, 1982.

4. Jensen, H. E., & Valentine, L. D. Development of the Enlistment Screening Test (EST) forms 5 and 6 (Tech. Rep. 76-42). Brooks Air Force Base, TX: Air Force Human Research Laboratory, May 1976.

5. Mathews, J. J., & Ree, M. J. Enlistment screening test forms 81a and 81b: Development and calibration (AFHRL-TR-81-54). Brooks Air Force Base, TX: Manpower and Personnel Division, Air Force Human Resources Laboratory, March 1982.

6. Vale, C. D., & Weiss, D. J. A study of computer-administered stradaptive ability testing (Res. Rep. 75-4). Minneapolis: University of Minnesota, October 1975.

7. Weiss, D. J. Strategies of adaptive ability measurement (Res. Rep. 74-5). Minneapolis: University of Minnesota, Dept. of Psychology, December 1974.

8. Weiss, D. J., & Betz, N. E. Ability measurement: Conventional or adaptive? (Res. Rep. 73-1). Minneapolis: University of Minnesota, February 1973.

9. Betz, N. E., & Weiss, D. J. Empirical and simulation studies of flexilevel ability testing. Minneapolis: University of Minnesota, July 1975.

10. Weiss, D. J. Final report: Computerized ability testing, 1972-1975. Minneapolis: University of Minnesota, April 1976.

11. Gorman, S. Computerized adaptive testing with a military population. In D. J. Weiss (Ed.), Proceedings of the 1977 Computerized Adaptive Testing Conference. Minneapolis: University of Minnesota, September 1977.

12. Pine, S. M. Reduction of test bias by adaptive testing. In D. J. Weiss, (Ed.), Proceedings of the 1977 Computerized Adaptive Testing Conference. Minneapolis: University of Minnesota, September 1977.

13. Betz, N. E., & Weiss, D. J. Psychological effects of immediate knowledge of results and adaptive ability testing. Minneapolis: University of Minnesota, 1976.

14. Prestwood, J. S. Effects of knowledge of results and varying proportion correct on ability test performance and psychological variables. In D. J. Weiss, (Ed.), Proceedings of the 1977 Computerized Adaptive Testing Conference. Minneapolis: University of Minnesota, September 1977.

15. Pine, S. M., Church, A. T., Gialluca, K. A., & Weiss, D. J. Effects of computerized adaptive testing on black and white students. Minneapolis: University of Minnesota, March 1979.

16. Jensema, C. J. Bayesian tailored testing and the influence of item bank characteristics. In Applied Psychological Measurement. Minneapolis, 1977, 1:1.

17. Moreno, K., Wetzel, C. D., McBride, J. R., & Weiss, D. J. Relationship of the Armed Services Vocational Aptitude Battery to three computerized adaptive subtests (Tech. Rep.). San Diego: Navy Personnel Research and Development Center. (In preparation).

18. DeWitt, L. J., & Weiss, D. J. A computer software system for adaptive ability measurement. Minneapolis: University of Minnesota, January 1974.

19. Sands, W. A. Development of a revised odds for effectiveness (OFE) table for screening male applicants for Navy enlistment (Tech. Note 76-5). San Diego: Navy Personnel Research and Development Center, April 1976.

20. Sands, W. A. Screening male applicants for Navy enlistment (Tech. Rep. 77-34). San Diego: Navy Personnel Research and Development Center, June 1977. (AD-A040 534)

21. Sands, W. A. Enlisted personnel selection for the U.S. Navy. A Journal of Applied Research, Spring, 1978, 31(1), 63-70.

22. Horner, S. O., Mobley, W. H., & Meglino, B. M. An experimental evaluation of effects of a realistic job preview on Marine recruit affect, intentions, and behavior. Columbia: University of South Carolina, September 1979.

23. Swanson, L., & Rimland, B. A preliminary evaluation of brief Navy enlistment classification tests (Tech. Bul. 70-3). San Diego: Navy Personnel Training Research Laboratory, January 1970.

WORD PROCESSING IN THE CLASSROOM: USING MICROCOMPUTER-DELIVERED SENTENCE COMBINING EXERCISES WITH ELEMENTARY STUDENTS

by Karen Piper

Texas Tech University
Texas Instruments Incorporated
Lubbock, Texas

Abstract

This article investigates the feasibility of using the microcomputer to deliver sentence combining instruction to upper elementary students. Background research on traditional sentence combining instruction is reported, and the use of the microcomputer in writing instruction is discussed. The author describes recent research in a small elementary school, which yielded positive results for microcomputer-delivered instruction in sentence combining and expansion activities. Positive results were evident in the areas of writing, revisions, attitude toward composition, attitude toward sentence combining instruction using the microcomputer, and student ability to effectively use word processing programs.

Introduction

As the microcomputer becomes more common in classrooms, language arts teachers will be less enchanted with it solely as a drill and practice delivery system. They will desire other uses for it that will increase its applicability to the total curricular program. One direction for the expansion of microcomputer use is in the area of structured writing instruction such as sentence combining.

Recent improvements in the quality and user-friendliness of word processing programs have already stirred an interest in the microcomputer as a tool for writing instruction. Yet the studies that examine the role of the microcomputer in writing most often deal with open, creative writing, not structured instruction in writing technique. Sentence combining instruction which uses the monitor as paper and the keyboard as pen can aid students in several areas, some of which include linguistic development and structure, comprehension of complex structures, and motivation to write and revise.

Research in the Classroom

In order to investigate some of the effects of microcomputer-delivered instruction in sentence combining, the author and Dr. Michael Angelotti, of Texas Tech University, designed a study using sentence combining treatments presented on the microcomputer to fifth graders in Abernathy Elementary School. Of particular interest to the researchers was the effect of using the microcomputer as a writing tool on the areas of writing motivation and syntactic maturity. Growth in syntactic maturity, a basis for writing ability and reading comprehension, might also foster gains in these important areas. To date, results from this study exceed expectations, especially in the areas of motivation to write and revise and writing improvement.

Sentence combining is usually presented in a large group, followed by oral work in small groups, and finally, with the student writing out his combinations. Using the microcomputer in this scheme has many benefits. Effectively used, the microcomputer should enhance teacher abilities to provide high quality instruction in sentence combining. Areas particularly affected by this treatment include writing motivation and syntactic maturity as evidenced by growth in writing ability and reading comprehension. The use of microcomputers as a delivery system in this task should: 1) allow for individualization of instruction with student self pacing, 2) enhance student motivation and interest, 3) enhance student awareness of the manipulative quality of language, 4) provide students with immediate feedback through print-outs and spontaneous interaction with video display, 5) motivate students to write, and 6) provide students with a record of student processing abilities.

The major goal of sentence combining instruction is an increase in syntactic maturity. Based on research in traditional sentence combining, this desired increase in syntactic maturity might result in an increase in reading comprehension. Word processing systems should be useful in teaching sentence combining because of the ease of text manipulation and print-out capabilities. However, this has not been previously investigated in regards to the feasibility of such instruction, the ease of implementation, the motivational aspect, and the overall effect of such instruction on student abilities and attitudes. The major goals of this research included determining

the feasibility and examining the total effect of this instruction on writing growth, reading comprehension, and attitudes toward writing. In order to provide necessary background for the use of sentence combining activities delivered by the microcomputer, the results of traditional sentence combining techniques will be reported. This information will be followed by a brief description of the study conducted by the researchers during the spring of 1983 and a discussion of some of the implications of this exciting technique.

Traditional Sentence Combining

Research in sentence combining supports it as an effective way to improve student writing and although the evidence is inconclusive, sentence combining also seems to enhance reading comprehension. Sentence combining theory features the use of embedding techniques to form more complex sentences. These techniques encourage students to embed modifiers, make deletions, and perform various transformations (O'Hare,1973; Strong,1976; Combs,1977). Sentence combining activities force students to use their linguistic knowledge to manipulate language. An additional benefit of sentence combining is that students are asked to embed kernel sentences, which forces them to keep longer discourse in their heads. This activity promotes chunking, a memory technique which can lead to improved organization and enhanced cognitive maturity (O'Hare,1973; Sternglass, 1980).

Due to the interrelationships among the language arts, sentence combining may improve reading comprehension as well as writing ability (Combs,1977). Encoding written language is not simply the process of writing down oral language. Written language is more grammatically complex, reflecting more embeddings. In addition, written language requires specific rules (i.e., punctuation and spelling) for correct implementation. Decoding written language (reading comprehension) is equally complex, involving student experiential, intellectual, and linguistic resources in relation to content, vocabulary, and sentence structure. A common denominator in these language processes is syntactic maturity. Researchers were initially interested in establishing a relationship between syntactic maturity and writing ability (Hunt,1965). Once established, they directed their attention to the relationship between syntactic maturity and more efficient reading comprehension. Some researchers felt that sentence combining might also foster reading syntactic maturity as evidenced by comprehension (Stotsky,1975; Ney,1980; White,1980).

Research by Evanecko, Ollila, and Armstrong (1975) indicated that two language competencies, fluency and control of syntactic complexity, underlie reading comprehension proficiency. They suggested that building these two competencies would improve reading comprehension. Stotsky (1975; 1982) reviewed studies that attempted to show that enhanced syntactic skills through writing activities improved reading comprehension. She confirmed that an increase in syntactic skills does effect reading comprehension (1977). The overall effects of sentence combining practices on reading are still inconclusive, and several researchers urge further investigation (White,1980; Stotsky,1982).

Sentence Combining with Microcomputers

Little research has been found that specifically addresses the use of microcomputers to deliver instruction in sentence combining for the purpose of enhancing the syntactic maturity and reading comprehension of school aged populations. Most of the research focuses on either college populations, or is restricted to creative writing. Results of microcomputer use in creative writing, however, provide a necessary foundation for the extension of writing instruction to sentence combining activities . For example, Schwartz (1982) stated that the use of a computer-based text editor encouraged greater manipulation of written material. Bradley (1982) reported positive results when students used word processing systems for creative writing. In addition, she compared student ability to use various word processing systems. Seymour Papert in Mindstorms (1980), reported that children at his MIT Computer Lab often went from "total rejection of writing to an intense involvement accompanied by rapid improvement of quality within a few weeks of beginning to write with a computer." Schwartz (1982) spoke of the positive effect of the microcomputer on student revisions. Educators and researchers are beginning to examine the feasibility of using the microcomputer in writing instruction and to suggest it as a possible delivery system for sentence combining instruction (Bradley, 1982; Cronnell, 1981).

The manipulative characteristics of composing with word processors, coupled with the motivational value of such a tool has great potential to tempt students to write more, revise more often, combine and embed language elements, and to increase in syntactical maturity as evidenced by writing ability. It seems plausible that as students are trained to deal with more complex written syntactical units, an increase in reading comprehension might occur. The microcomputer seems to be a reasonable delivery system for instruction of this sort, and can help teachers more adequately incorporate instruction in sentence combining into their classrooms. One way sentence combining instruction delivered by the microcomputer can be implemented is the method used in this study. Use of this technique or a related procedure can help the teacher expand the microcomputer from simply an instructional tool to an instructional medium which holds great promise in the area of structured writing technique.

The Research Technique

The purpose of this research was to sift through several notions concerning the feasibility of sentence combining exercises and sentence expansion activities delivered by the microcomputer. Of specific interest were growth in syntactic maturity in the writing and reading comprehension realms, the motivational quality of sentence combining instruction delivered by microcomputer, and student attitudes toward the technique.

Due to funding constraints, microcomputer availability, and a desire to investigate the feasibility of large-scale implementation of this technique, a series of case studies (N of 1 design) was chosen. Several case studies of students representing different ability levels were needed to help determine the overall effect of these sentence combining exercises. Fifth grade students comprised the population. Student scores on the Comprehensive Test of Basic Skills (CTBS), language and reading subtests, and ethnicity were used in drawing a random sample of matched pairs (names were masked). The control and experimental pairs were chosen to reflect four students who were above grade level in their scores and four students who were below grade level in their scores. Student pairs were then randomly assigned to either the control or experimental group, so that each group was composed of two above average and two below average students.

Pretesting and posttesting consisted of a timed writing sample, a cloze-type reading comprehension test, and a semantic differential attitudinal measure designed to reflect attitudes toward: school, writing assignments, writing revisions, reading, and the use of microcomputers for instruction.

The Treatment

Sentence combining lessons and exercises were presented in forty-five minute segments twice weekly by the researcher. As a group, students discussed each new technique with the researcher and later with a student partner. Each student was asked to do the lesson on an Apple II Plus (48K) microcomputer, using the Apple Writer word processing program. Strong's method of open-style cluster presentation was used. His text, Sentence Combining and Paragraph Building (1981) served as a methodological model for the exercises. After the students had completed each combination and typed in "writeouts" (combinations), they were asked to expand the group of writeouts into a story.

The second lesson of each week served as a review of the sentence combining technique presented earlier and a chance for revision of student stories. Each student received a print-out of his combinations and resulting story. Student editing teams (a low ability student paired with a high ability student) read and edited each other's work. Each author made his own revisions on the microcomputer. Final print-outs were obtained and shared with the group. Sharing written stories was a special time, and the pride of student achievement was evident.

Conclusions

Due to the nature of small group or case study research, few statistical measures were employed, and the reader is cautioned against generalizing the results of this study. However, general conclusions can be drawn relative to this study and graphs of individual student achievement will be presented at NECC.

While further research in this area is needed, it appears that this method of writing instruction can be used successfully with upper elementary students. Although six weeks of sporadic treatment may not be long enough to draw final conclusions, initial results look positive. Fifth grade students in this study were able to quickly learn both the word processing system chosen and the sentence combining techniques. Students in the group that used microcomputers tended to show a gradual increase in semantic maturity as measured by T-units, or units of the "shortest grammatically allowable sentences into which written language can be segmented (Hunt, 1965)." In addition, students in the experimental group were observed using structures not seen in the writing of the control group (i.e., appositives, many complex sentences, and items in a series). Each of these structures had been presented in sentence combining exercises. A final analysis of the results of reading comprehension growth is pending and will be presented at NECC.

Perhaps the most notable change for the experimental group and the greatest difference between the groups was in the attitudinal realm. Attitudes toward writing and writing revision using the microcomputer were very positive. Teachers visiting the computer center were often surprised by the intent with which the students tackled the sentence combining activities. Even the two students previously lableled as "below average" showed great enthusiasm for the project and were motivated not only to complete the sentence combining activities, but also to revise and expand what they had previously written. Another indication of the positive responses to the sessions was that students revised all aspects of their work, including content, punctuation, grammar, and even spelling. Student interest was high, and they were extremely reluctant to leave the sessions. This group viewed with disdain the sessions at which both groups (control and experimental) were asked to give written samples using pen and paper. In other words, they vigorously preferred using the microcomputer as a writing tool over pen and paper. An investigation of attitudinal ratings toward school, writing instruction, writing

revision, using the microcomputer, ease of use of the word processing system, and sentence combining exercises was very encouraging.

It appears that not only is sentence combining delivered by microcomputer effective, but it is also a popular form of instruction. This added incentive encouraged students to write, revise, and read what they and others had written, making them particularly open to instruction. The researchers feel that the conclusions of this study and the related implications of these findings can be very impactful on language arts instruction and future research.

Implications

Although final results are pending, this research indicates several positive effects of the use of microcomputers for sentence combining instruction. The motivational aspect in and of itself might prove highly useful in countering the negative feelings many students hold toward writing and revisions. Students who wrote with the microcomputer tended to view the instruction and the writing positively. Their stories were generally more complex than their matched pair in the control group. They enjoyed the sentence combining activities, and by the end of the second treatment phase they were able to use fairly advanced combinations without prompts from the researcher. Other positive side effects were evident. Absenteeism was non-existent throughout the study. Students became more cognizant of how to dissect and read complex structures because they knew how to 'build' them. Proficiency with the word processing system increased quickly and the students became much more efficient in their use of microcomputer time. At this point they were able to concentrate more intently on their writing.

While enthusiastic about these results and the implications they hold for language arts education, the researchers recognize that they have simply scratched the surface. The possibilities for the expanded use of microcomputer instruction in reading and writing are exciting and ever increasing. Word processing in the classroom is not a panacea, but it does offer opportunities for growth in the language arts processes. The expansion of the pilot study described here that is planned for the fall will help lay the groundwork in this investigation, and far-reaching results are anticipated. However, these studies can only serve to provide a knowledge base and spur interest in this area. Further research needs to be done so that we may fully investigate the use of word processing microcomputers as tools for structured writing and reading instruction. Teachers and researchers must be tempted to explore the applications of such an instructional program so that students might more fully realize the vast educational possibilities offered by the word processing microcomputer in the language arts classroom.

REFERENCES

BRADLEY, Virginia. Improving students' writing with microcomputers. Language Arts, 1982, (October),59,7.

COMBS, Warren. Some further effects and implications of sentence-combining exercises for the secondary language arts curriculum. Unpublished doctoral dissertation,University of Minnesota, 1975.

COMBS, Warren. Sentence-combining practice aids reading comprehension. Journal of Reading, 1977, 21,1,18-24.

CRONNELL, Bruce, et.al. Using microcomputers for composition instruction. Paper presented at the Conference on College Composition and Communication (Dallas, Tx., March, 1981).

EVANECKO, Peter, et.al. An investigation of the relationships between children's performance in written language and their reading ability. Research in the Teaching of English (1975),8, 3, 315-326.

HUNT, Kellogg. Grammatical Structures Written at Three Grade-levels. National Council of Teachers of English, Urbana,Illinois, 1965.

NEY, James W. A short history of sentence-combining: Its limitations and use. English Education, 1980, 11, 3, 169-177.

O'HARE, Frank. Sentence-combining: Improving Students' Writing Without Formal Grammar Instruction. Research Report No. 15, National Council of Teachers of English, Urbana, Illinois, 1973.

PAPERT, Seymour. Mindstorms. Basic Books, Inc.: New York, New York, 1980.

SCHWARTZ, Mimi. Computers and the teaching of writing. Educational Technology, 1982, (November), pp. 27-29.

STERNGLASS, Marilyn. Sentence-combining and the reading of sentences. College Composition and Communication, 1980, 31, 3, pp. 325-328.

STOTSKY, Sandra. The role of writing in developmental reading. Journal of Reading, 1982, 25,4,pp. 330-340.

STOTSKY, S.L. Sentence-combining as a curricular activity: Its effect on written language development and reading comprehension. Research in the Training of English, 1975, 9, pp. 30-71.

STRONG, William. Close-up: Sentence-combining. Back to basics and beyond. English Journal, 1976, 65, 2, pp. 56-64.

WHITE, Regine, et.al. Reading, writing, and sentence-combining: The track record. Reading Improvement, 1980, 17, 3, pp. 226-232.

THE COMPUTER IN THE WRITING CLASS: PROBLEMS AND POTENTIALS

C. Daiute, P. O'Brien, A. Shield, S. Liff, P. Wright, S. Mazur, W. Jawitz

Teachers College, Columbia University

Writing teachers have recently become enthusiastic about the use of word processing for improving their students' writing. This paper offers guidelines for introducing computers in the writing class. We note preliminary results of our research on the effects of computers for developing writing skills, but our focus is on the important practical considerations that make the computer a useful writing tool rather than a problem.

Janie

Draft

> This may sound pretty weird and sad but sometimes I dream that the whole world would blow up. Another person and I would be the only survivors and we together would start the world all over again. The other person and me would rule the world. We would learn all the technology we need to know, how to fly and airplanes, and all that stuff. We would also learn to care for eachother. I think I would rule the world better than the president rules the United States of America, I hate to say it but its what I think.

Revision

> This may sound weird but I dream that the whole world would blow up. Another person and I would be the only survivors. We would start the world all over again. We would rule the world together. We would also care for eachother a great deal. We would learn the technology we need to know, how to fly airplanes, and all that stuff. I think I would rule the world better than president Regan rules the United States. I hate to say it but its what I feel.

Janie began to improve drafts of her texts when she wrote on the computer. Unlike most eleven year olds who have had extensive practice revising their writing, Janie made quite a few significant as well as superficial changes in the piece above. Janie did not simply read her text. She saw it more objectively than when she had first written it and read it over thinking "it was fine." She deleted words, moved sections, changed words, and added details, such as the President's name. She formed a new, more focused piece.

Revising behaviors like Janie's are not typical of most children. Using computers can stimulate children's revising activities, which many writing teachers feel are extremely important. Nevertheless, using computers is not as easy as using other writing instruments in the classroom.

The authors of this paper are all working on the Computers and Writing Project at Teachers College, Columbia University, but we have varied backgrounds and varied roles. We are researchers, teachers, and students who have used computers for our own writing. Our research has taken place in three school settings in Manhattan: at J.H.S. 118, the United Nations International School, and Teachers College in the Department of Communication, Computing, and Technology in Education.

Our overall goal of the research is to find ways to make writing easier for children and adults alike. Our specific goals differ. The researchers on the project are focusing on the ways in which computers relieve physical and cognitive burdens that affect young writers as they compose and revise. The teachers, however, are more interested in methods of implementing computers in their writing classes to free student writers to express their ideas more easily than they can with traditional writing tools.

The purpose of this paper is to suggest why computers may be valuable tools to use in the

writing class and to offer specific suggestions on implementing them as writing tools. Using computers in the writing class takes money, time, and patience, so we would like to offer guidelines. In addition to presenting the researchers' and teachers' reasons for using computers, we discuss the logistical problems involved; we offer solutions to these problems by presenting our methods for overcoming them. Finally, we discuss preliminary results of the effects computers have on the writing of school children.

Research Goals

Our research goals have been to explore the cognitive causes of writing difficulties, especially revising. The computer seemed the ideal writing instrument to use because it offers the possibility to reduce certain physical and cognitive burdens that make revising difficult. Computer word processing programs allow writers to type text into the computer, and make small or large changes by giving commands rather than by recopying. The program automatically incorporates each change into an updated, neatly-formatted version of the text. Thus, the dread of recopying does not discourage children from revising.

Word processing programs are also useful tools for stimulating revising because they can be augmented with automatically-presented comments that can guide writers in self-questioning strategies, modeled on those that students have with teachers and peers. Conferencing -- discussing texts with student writers -- has been found an effective method of instruction (Calkins, in press). We thought some form of conferencing on the computer would be helpful because the writer would initiate it and would thus be in charge of the self-monitoring and rewriting processes. We were, in short, using the computer to show children how writers talk to themselves to improve their texts. The program we developed to present young writers with prompts to stimulate revising is called Catch.

At any time during or after writers compose with the word processing program, they can give a command to see a list of analyses and prompts Catch gives to guide writers as they revise. If a writer, for example, selects the "empty word" option, the program identifies unnecessary words such as "sort of" and "well." As these words are highlighted on the monitor, a prompt appears at the bottom of the screen. The prompt that appears with "empty words" is "The highlighted words may not be necessary. Can you make changes?" The writer can immediately make changes if he or she wishes. Writers using the program are aware that it may identify words that are empty in some contexts such as "well" in "Well, I will begin with my childhood," but not in others, such as "My first memories are of throwing pennies into a wishing well."

Getting Started

Teachers have expected that the dynamism and precision of writing on the computer would encourage their students to reconsider the purpose and style of their papers when they revise because they would not have to recopy, as well as to pay closer attention to details of spelling and grammatical mechanics. The following quote by Peggy O'Brien of the United Nations International School expresses some of teachers' expectations.

> As soon as I felt my own response to using the computer as a writing tool ...I was on the look out for a way to use computers in my English classes. When I discovered that you could do word processing with micros, I made every effort to get hold of one. The big "pay off" I expected was the reinforcement of all English skills. I theorized that there would be no stopping the kids who already liked writing and kids whose slowness or lack of coordination kept them away from expressing the fantastic ideas they had, or who hated the messy page they always ended up with, would particularly benefit from using computers. I also suspected that introducing the students to the range of options available from reformatting would dramatically affect their written work.

Ms. O'Brien initiated the process on her own:

> Right from the beginning, I received strong encouragement, support and involvement in future planning from the Dean of Studies at the school. I knew we had machines in the school which were underutilized. By a certain amount of conspiring and begging, I managed to get the TRS-80 for use in my classes and persuaded the school to buy Scripsit and a trolly to make the thing mobile. That got us started. We used the machine without a printer for several months, putting pressure on all the people we could to help us to get a printer. In the meantime, I was encouraged by the Dean of Studies to write a proposal for an Apple II plus with disk drives, printers, and monitors. I proposed teaching a course in language enrichment (making a grademagazine) for the new academic year which entailed the use of all the technology available in the school, and so managed to get my hands on the Pets temporarily. Funding has been done from the school's overall budget.

On the other hand, school administrators often present the impetus for encouraging teachers to be part of the technological age. Arthur Shield of J.H.S. 118 describes the process:

> The principal of my school asked me to teach a word processing course. The principal informed me that through the Division of Curriculum and Instruction of the New York City Board of Education, our school would acquire five micro-computer systems, to be specifically utilized for the improvement of writing skills. These systems would be purchased with New York State Umbrella Funds. The Principal was concerned about the poor writing ability of students and the traditional writing course had not been successful. Initially, I was against teaching such a course for two reasons. First, I recognized that for me change is difficult and adjusting to a different set of conditions would be quite arduous. Secondly, I felt intimidated by computers. I had no previous experience with computers and further realized that within a bureaucracy there would be little opportunity for me to adequately learn the intricacies of the hardware, let alone the software for word processing. On the other hand, I was ready and even anxious to undertake new teaching objectives.

Logistical Problems and Solutions

Managing Computers and the Writing Class

Managing the computers, human and intellectual energies, and curriculum in the classroom is a major job. Arthur Shield has five Apple II plus machines with disk drives and one printer. His comments highlight the major problems and suggest ways around them.

> One must be extremely organized in order to be successful with the program and having only five computers and approximately twenty-five students in the classroom has created various problems. There is never enough preparation time. I have to prepare lessons so that those students who are not on the computers are actively involved in a motivating activity. Since the majority of the students had had absolutely no exposure to computers, I had to teach the basics. A typical class period involves my running back and forth within the classroom to deal with questions, bugs, etc., that continually arise. I have had to interrupt several lessons at crucial points in order to answer a question of the students working on computers. You also have to make sure that you have adequate supplies, such as printer paper and diskettes, essential ingredients for any word processing program. The teacher also has to have contingency plans for times when one of the machines does not function. In order to expedite the process of servicing the microcomputers, I had to learn whom to contact within the company when problems arise.
>
> Finally, knowledge of the hardware is absolutely necessary. Often times, I had to determine which piece of hardware was malfunctioning and needed servicing. When dealing with computers, the guiding precept is to become as knowledgeable as possible, which in turn tends to avoid the waste of time. Students have been learning to become independent when using the computers. The goal is to have the students depend less upon the teacher when dealing with the practical matters relating to the computer.
>
> I strongly recommend that a teacher who undertakes computer instruction be well-prepared and well-versed regarding the programs that are selected for the word processing course. Access to a computer on a regular basis is fundamental in order to master the available software. Registering for college courses dealing specifically with computers, particularly those related to the field, is vigorously recommended.

Training Teachers

Teachers must be comfortable with all the tools their students use in the classroom. We have found that the teachers who use word processing programs for their own writing before they introduce them in the classroom travel the smoothest road. First, adults and children have different styles of approaching and mastering computing tools, so they should probably not have their first experiences at the same time (Daiute, 1983). Second, teachers who use word processors themselves understand the benefits and drawbacks and often dream up innovative applications to their writing curriculum as well. Finally, teachers who have used programs extensively, know the quirks, bugs, and ways around them - which occur in many programs at this early stage in the state of the art. Teachers who intend to use computers in their writing classes should thus spend several months training and practicing.

Training Students

In this project, research assistants helped to prepare materials and to train the students in using the editor. Research assistants helped in copying the required number of diskettes, preparing and copying necessary writing material, and helping the teacher to organize a large number of students onto the limited

number of machines. For example, it was important that the assistants work in conjunction with the teacher to ensure that two students in the same class were not using the same diskette.

The research assistants also helped train the students to use the word processing program. They were available to answer any questions that the students might have, and to deal with any system crashes that might otherwise have frustrated a new learner.

Of course, classroom teachers do not have college or graduate students available to assist them. This need not present a problem, for the students themselves act as assistants. We noted from our training session that students who were familiar with a word processing program, or who had had previous experience on computers, were very anxious and willing to help their fellow classmates who were having problems. A typing tutor program was developed to familiarize students with the keyboard. Students were encouraged to develop their typing skills since this would facilitate their writing on the computer. Students were exposed to the typing program for several weeks before being introduced to the word processing program, and although students did not learn to touch type, most students felt comfortable with the keyboard when the word processing program was introduced. Students who were not comfortable with typing were encouraged to continue using the typing tutor program.

A text containing errors in spelling, word repetition, word omissions, and organization was copied onto the students' diskettes. A step-by-step worksheet was created to lead the students through several stages from simple machine operation to word processing commands. The worksheet introduced commands which ranged from the deletion of one character to the more complex movement of a block of text.

Effectiveness

Students worked in groups of 3-5 at each computer for a 35 minute session. This apprach was chosen to allow students the freedom to experiment within a structured framework. We used one practice assignment to introduce the students to the word processing commands. Our training methods appeared to be more effective for the higher grade levels (i.e. 8th and 9th) than for the lower grade (i.e., 7th). This seemed to occur because the lower grades had larger classes, resulting in less computer time for each student. There was also a noticeable difference in computer -student interaction. The lower grade students often executed commands mechanically without regard to function and they rarely watched the screen when correcting errors.

Groups were selected at random and asked to work together on the lesson. At first, students were dependent upon the teacher or assistants for a quick solution to any difficulties that arose, such as forgetting what key to press for a deletion. However, if the instructors were not available, the students would refer to their worksheets and find their own solutions.

Eventually, students called upon other members of the group to collaborate and work toward their own solutions. More group interaction was observed among students in the higher grade levels, again perhaps due to the smaller group size. It is important to provide each student sufficient time to work on the computer.

Using the Computer for Meaningful Writing Activities

Children need meaningful topics and composing strategies so they can benefit from the physical and cognitive aids offered by the computer writing tools. It is always important to present children with good writing topics to enhance the development of writing skills. It is especially true when they are learning to write on the computer. It is particularly important to present a topic that is intrinsically motivating because writing on the computer is difficult at first. There are many new things to keep in mind, and if the child must dwell on an uncomfortable topic or style, the writing task easily becomes overwhelming.

The child who is interested in the topic is much more likely to stick at it longer, produce more, and master the technique. Good topics seem to be those that have the child write in an informal style about something he or she feels strongly about. Opinion essays such as "Why I like/dislike New York City" are good. This kind of topic seems to provide enough structure, still allowing each child choices about content and style.

Other writing tasks that work well are autobiographies or stories about the child's group of friends. Most children will write comfortably on those two, but for some children they are too personal. When a child shows signs of such discomfort, allowing him to invent a fictional character or characters usually solves the problem.

Effects of Computers on Writing: Preliminary Results

Our preliminary findings about the effects of computers on the children's writing skills are based on classroom observations and analyses of the children's texts. An observer asks a student for permission to watch while the student works on the computer. Occassionally, a student declines the request to be observed, but most are clearly happy to display their work. The observer has a list of observation points from the student's overall enthusiasm, to his or her typing skills, and from the types of revisions made, to the overall quality of the text.

Our preliminary observations confirm three crucial elements in bringing children and computers together. First, and in a sense most importantly, the students have an overall positive attitude about the computer. They like the idea of having

computers in their classroom, and they are excited about using them. For whatever reason, their excitement serves to diminish the more frustrating aspects of working on the computer. Namely, the sheer mechanics of using a computer presents some problems for children. For example, we found that the students who type better (regardless of method), seem to be more generally enthusiastic about writing on the computer. Also, actually starting the writing process can be tricky because of preliminary steps required to create and maintain a file, such as finding, writing, saving, and transferring text from the diskette.

It is also clear that at least in the early stages of writing on the computer, students favor the less abstract word processing commands over the more abstract and powerful commands. For instance, though they are aware of commands that move the cursor by a line, sentence, or even a paragraph at a time, the large majority of students move the cursor a character at a time for sixty or seventy strokes to get to a word they want. to correct in the previous sentence or paragraph. However, it is important to remember during this first and often frustrating stage, that as students become more familiar with the keyboard, hardware, and editing commands, they more frequently experiment, and become comfortable with the higher editing commands.

Our preliminary experimental results suggest that the word processing program and Catch generally stimulated children to revise.

Most of the children did more revising with pen after they had used Catch for about four weeks. Janie, however, was one of the children who did more revising with the computer, but did not do as much when she used pen after having used the computer and Catch as a guide.

We have several interpretations of these results. Weak writers like Janie may find the new tool makes writing more comfortable. Often weak writers simply have not read or written very much, so they have not developed a style and do not have text models to imitate. Children with strong writing skills, on the other hand, who have written extensively have developed a sense about writing. Such writers have also become comfortable with a traditional writing instrument, usually the pencil or pen. The computer may offer them less relief as a writing instrument until they are as comfortable using it as they are with pencil or pen -- the instrument they had been using all along. Thus, initially the computer may help them less than it does children who had not become comfortable with other writing instruments, and it takes longer for experienced young writers to benefit greatly from its capacities. Similarly, good writers bring composing and revising strategies to their first use of the computer, so prompts like those in Catch may at first have less impact.

Effects of the computer depend on the writer's abilities. Even though the most skilled young writers approach the computer with excitement, they find the mechanics of word processing somewhat frustrating. They sometimes feel that writing with pen and paper is easier and more reliable. Usually, enough of the advantages of word processing are apparent to help them pass this phase. Once they master the skills of interacting with the computer, the tool helps them write and revise efficiently.

Less skilled writers tend to come to the computer with the view that writing is a laborious process. Their attitude towards writing is generally negative and the computer does much to improve it. They also approach the computer with excitement and their intitial excitement never seems to wane.

In contrast to the majority of students, many teachers approach computers with fear. They feel that many of their students already know more about computers than they do. They learn to use the machines and decide how they will be most beneficial to their students' education. But, when they start with the computers in the classroom, they often find the extra logistic chores overwhelming. Nevertheless, once the major classroom management problem is solved, they become excited about the fact that their students find writing to be fun.

References

Calkins, L.M. Lessons From A Child: On the Teaching and Learning of Writing. Forthcoming.

Daiute, C. Computers and writing. Reading,Mass.: Addison-Wesley Publishing Co., in press.

A HYBRID HUMANITIES APPLICATIONS COURSE: "THE COMPUTER IN LITERATURE" AND "LITERATURE IN THE COMPUTER"

by Rudy S. Spraycar

Data Processing Department
United States Fidelity and Guaranty Company
Baltimore, Maryland 21203

Abstract

This paper reviews the development of an interdisciplinary course whose aims were threefold: 1) to use science fiction and non-fiction as vehicles for discussion of the computer's impact upon society and societal attitudes toward the computer; 2) after reviewing programming, to teach computer-aided techniques of literary analysis; and 3) to require the class to undertake team projects in analyzing language or literature with the computer; most teams included students both a) familiar with the computer but innocent of literary analysis and b) familiar with literature but naive about the computer.

Introduction

A curriculum task force report prepared on the Louisiana State University campus a few years ago noted the growing need for disciplines besides engineering and computer science to offer courses about the computer. In particular, the social impact of the computer needs to be addressed from a variety of disciplinary perspectives. Having taught an English department course in science fiction at L.S.U. for some years, I had noticed how effective science fiction is in stimulating class discussion about the man-machine interface in general, and about man's relations with the robot and the computer in particular. Thus it seemed to me that an English department course in computing ought to include a unit of readings in science fiction and selected non-fiction works that examine 1) the computer's impact on society and 2) the panoply of current attitudes toward computers. This portion of the experimental course that I developed and taught at L.S.U. in the fall of 1981 was entitled "The Computer in Literature."

On the other hand, there has been great interest in the application of the computer to the development of traditional tools for literary research, such as concordances, indices, and bibliographies. At first, however, only a few pioneers recognized the machine's potential to extend radically the application of quantitative and statistical research methods to problems of authorship and attribution and to fundamental research into the character of literary style. In many respects, this field is now approaching maturity and is a suitable subject for graduate and undergraduate instruction. The segment of this course entitled "Literature in the Computer" surveyed current applications of the computer in literary analysis and the preparation of literary research tools; the course culminated in student team projects in this area.

Background

The gradual acceptance over the past decade or two of science fiction as legitimate English curriculum content needs no rehearsal here. I would only note that the originality of this course consisted largely in its hybrid approach, stressing 1) the mutual interaction of computers and literature, and 2) successful communication between devotees of both literary and computing cults within a single classroom.

Slower initially, but recently booming, has been the recognition on the part of humanities scholars at large of the value of instructing students in computer methods for literary research. To be sure, such pioneers as Robert L. Oakman[12-15] recognized as early as 1968 the need to teach such research methods, as distinguished from the use of the computer in computer-assisted instruction (CAI), which has encountered less pedagogical resistance, owing to the celebrated crisis in the teaching of English composition.

The recent publication of textbooks in computer-aided literary research by Oakman[16] and Susan Hockey[10] has stimulated interest in the area. Nevertheless, the recent third edition of Robert D. Altick's[1] very fine and otherwise comprehensive textbook on literary research methods devotes but five pages exclusively to computer methods, despite the dust jacket's promise of the book's expanded treatment of such methods. If resistance to interdisciplinary research methods remains among humanists, perhaps a hybrid course like the one described here may prove a reasonable effort to bridge the gap of understanding.

Course Structure

The readings included two novels, several short stories, an anthology of computer-related fiction and non-fiction, and two textbooks on programming and literary computing (see Course

Materials, below).

The Computer in Literature

Non-fiction and science fiction materials were used to define the man-machine relationship. The computer's inpact on society was treated both in terms of changes in the life of the individual and broad social and public policy issues. Of course, these two levels of the problem cannot, in the end, be separated: such issues as privacy, whether jobs are created or lost (with some reference to the parallel issue of robotics), and military, social, and industrial impacts are often of public consequence precisely because of their effects upon individuals. The non-fiction readings stressed the present potentials and possible threats posed by information banks, networking, and such; the science fiction readings tended to force speculation about the future by extrapolating one or another trend to its logical extreme.

Ironically, the readings in science fiction, often obsessed with technology, provided some of the best examples of the paranoid view of the computer. As is the case with robots, computers are more threatening the more they are perceived as human-like, for through the ages what is almost, but not quite, human has often been regarded as monstrous. Out of the critique of myths about computers emerged an understanding of both the limitations and the true potentials of the computer, leading into a brief account of how the machine works.

Literature in the Computer

Literary data processing was thus introduced as an object lesson in how the computer can be used. The course then took up in some detail the problems in literary research that have proven most amenable to computer assistance. Computer-aided instruction (CAI) was treated briefly, using as an example the course authoring and instructional system that I designed, wrote, and tested at L.S.U., CALAIS (Computer-Assisted Language Arts Instructional System).[19-20] Finally, we glanced briefly at the frontiers of both computer-assisted statistical analysis of literary style and artificial intelligence. Throughout the first two units of the course, each student presented an oral report on some aspect of either social attitudes toward the computer or computer-aided literary research. The students then formed research teams to define problems in literary analysis and wrote programs to solve them.

Synthesis

The study and discussion of two science fiction novels about the computer (while the teams worked on their projects) provided an opportunity to synthesize the main themes of the course and to sort out myth and fantasy from the issues surrounding the relation between computer and society. In particular, the students attempted to decide whether the computer is a monster, a friend, or simply a tool.

The research projects brought together two kinds of students: those without previous experience with the computer helped to define the problem, design an algorithm, and participate in data entry when necessary; those who had suitable prior experience developed the program, generally receiving more credit for their participation in the projects. The team's joint development of the approach served two purposes: 1) it provided a sound basis for demystifying the computer in order to critique social myths about it; and 2) it gave both the knowledgeable and uninitiated students an opportunity to be a part of a "user/analyst" relationship. The latter experience may have been the most valuable portion of the course for the computer science majors who enrolled.

Course Materials

Several science fiction anthologies devoted to fiction about the computer were out of print. The Van Tassel anthology[21] offers a wide range of fiction and non-fiction, but it is largely out of date. Students and instructor alike, however, supplemented this anthology with up-to-date articles culled from current popular magazines and newspapers; many of these materials were presented in the course of the oral reports.

Particularly useful were two science fiction stories about the survival of man and the computer after a nuclear holocaust: Roger Zelazny's "For a Breath I Tarry,"[22] a charmingly humorous, optimistic view, and Harlan Ellison's "I Have No Mouth, and I Must Scream,"[7] a fiendishly pessimistic vision.

A good introduction to programming and the concept of the algorithm was provided by Richard Conway's Programming for Poets: A Gentle Introduction Using PL/I;[3] the book is also available in versions using Basic, Fortran, and Pascal.[4-6] I chose PL/I because it is a relatively widely implemented language, especially at academic IBM installations, and is amenable to string processing. (However, see discussion below of revision of the course.)

Both Oakman's[16] and Hockey's[10] texts on literary computing are excellent; because my institution had an IBM system, and Oakman's book is somewhat IBM-specific, I chose it.

Finally, many science fiction novels would serve well for the final synthesis of the course. I chose Heinlein's The Moon Is a Harsh Mistress[9] and Brunner's The Shockwave Rider.[2] The former at once humanizes the computer and characterizes it as a tool that has become a friend. The latter dramatizes the potential hazards of widespread networking, then falls back upon the conventional moral position that the final responsibility for man's tools and his social organizations, especially bureaucracies, always rests with the individual; here a brave hero saves mankind from a bureaucratic behemoth objectified as a computer network. Other

science fiction novels could provide grist for biases other than my own, of course.

Syllabus

I append a course outline; for Conway's PL/I, one might substitute Snobol4.[8]

Student Projects

One group compared samples from Robert Heinlein's novels, written between 1939 and 1980, in terms of twenty-two variables. Significant changes were observed: sentence length increased over the years, as did the proportions of the words in the text that were enclosed within quotation marks. The students interpreted the latter feature as a sign of Heinlein's improved ability or desire to characterize through what a character said rather than through authorial comment.

Another team drew five samples of journalistic prose written in 1961 from the Brown University Standard Corpus of American English;[11] each word in the Corpus has already been labelled or tagged grammatically. The group confirmed the professional rule of thumb that average word length in newspapers is about five letters (their samples ranged from 4.7 to 5.1 characters per word). The group failed to find significant patterns in distribution of parts of speech either between samples or within an individual news story as a function of "pyramidal" journalistic style.

A third group interested in computer-composed poetry rejected a purely random approach to word selection in favor of a program with conversational natural-language prompts for parameters of verse form, theme, end-rhymes, and so forth. Finally, one student wrote a concordance-generating program, and another worked at debugging and refining the EYEBALL literary analysis software package, written in Fortran,[17-18] for local use.

Student Response

The students were asked on a questionnaire to rank each component of the course on a scale of one (must keep), two, three (neutral), four, and five (drop it). The results were as follows:

Table 1. Student Questionnaire. Averaged Rankings of Course Materials.

Social Issues	1.57
Van Tassel	2.42
Science Fiction Short Stories	1.67
Heinlein Novel	2.33
Brunner Novel	2.90
Computer Methods	2.33
Oakman	2.80
Conway	3.00
Oral Reports	2.73
Special Projects	1.73

Most popular (indicated by the lower numbers) were the social issues topic, the science fiction short stories (not the novels), and the special projects, in that order. All the other items were ranked favorably except the Conway book, about which the students were, on the average, neutral.

This neutrality was reflected in the students' responses when they were asked whether they would have preferred a standard programming text: yes, 50 percent; no, 43 percent; not sure, 7 percent. 49 percent favored substituting for PL/I a string-manipulation language, Snobol4, while 20 percent disagreed, and 33 percent were uncertain.

Finally, 93 percent of the respondents favored the addition of a one-credit laboratory period for instruction in programming and a start in hands-on experience for those who needed it.

Revision of the Course

Unfortunately, the only course number available in my department for such an experimental course was at the sophomore level; of the sixteen students who completed the course, eleven were upperclassmen. I recommended that the course be formally instituted at the senior level, with enrollment open to graduate students. While most of the projects were completed satisfactorily, the aims of the special team projects outlined above would be better achieved if all the students brought to the course more special competence in either programming or potential applications, e.g., literature, music, etc. Students without prerequisite courses in computer science should be required to enroll in the one-credit laboratory that the students favored so strongly. The assignment of science fiction readings as an occasion for discussion of social issues should be retained, as should the oral reports and, above all, the special team projects.

Appendix

Course Outline: Literature and the Computer

Week	Readings
	I. The Computer in Literature
1	Introduction
2	Van Tassell, sections 1-4
3	Van Tassell, sections 6-9
4	Science Fiction Short Stories
	II. Literature in the Computer
5	Oakman, chapters 1-2
6	Oakman, chapter 3; Conway, part I, chapters 1-2
7	Conway, part I, chapters 3-6
8	Conway, part I, chapters 7 & 9; mid-term exam.
9	Van Tassel, section 5; Conway, part II, chapter 1
10	Conway, part II, chapters 2 & 4; Oakman, chapt. 4
11	Conway, part II, chapter 5; Oakman, chapters 5-6
12	Oakman, chapters 7-8
	III. Synthesis
13	Heinlein, The Moon Is a Harsh Mistress
14	Brunner, The Shockwave Rider
15	Results of Special Projects; Conclusion

References

1. Altick, R. D. The Art of Literary Research. Third ed., rev. J. J. Fenstermaker. New York: Norton, 1981.

2. Brunner, J. The Shockwave Rider. New York: Ballantine, 1976.

3. Conway, R. Programming for Poets: A Gentle Introduction Using PL/I. Cambridge, MA: Winthrop, 1978.

4. Conway, R. and Archer, J. Programming for Poets: A Gentle Introduction Using FORTRAN, with WATFIV. Cambridge, MA: Winthrop, 1978.

5. Conway, R. and Archer, J. Programming for Poets: A Gentle Introduction Using BASIC. Cambridge, MA: Winthrop, 1979.

6. Conway, R., Archer, J., and Conway, R. Programming for Poets: A Gentle Introduction Using PASCAL. Cambridge, MA: Winthrop, 1980.

7. Ellison, H. "I Have No Mouth, and I Must Scream," in Silverberg, R., ed., The Mirror of Infinity, New York: Harper and Row, 1973.

8. Griswold, R. E., Poage, J. F., and Polonski, I. P. The SNOBOL4 Programming Language. Second ed. Englewood Cliffs, NJ: Prentice-Hall, 1971.

9. Heinlein, R. The Moon Is a Harsh Mistress. New York: Berkley, 1968.

10. Hockey, S. A Guide to Computer Applications in the Humanities. Baltimore: Johns Hopkins, 1980.

11. Kučera, H. and Francis, W. N. Computational Analysis of Present-Day American English. Providence, RI: Brown University Press, 1967.

12. Oakman, R. L. "Computer Methods for Humanities Research: An Interdisciplinary Approach at South Carolina,"Proceedings of the IBM Symposium on Introducing the Computer into the Humanities, June 30-July 2, 1969, Poughkeepsie, NY: IBM, 1969.

13. Oakman, R. L. "Computers for the Humanities," Humanities in the South: Newsletter of the Southern Humanities Conference, 35 (Spring 1972), 4 & 10.

14. Oakman, R. L. "Computer Education for the Humanities: Multiple Possibilities at the University of South Carolina," Proceedings of a Fourth Conference on Computers in the Undergraduate Curricula, June 18-20, 1973, Claremont, CA: The Claremont Colleges, 1973.

15. Oakman, R. L. "A Videotape Course for Computer Education in the Humanities," Computers and the Humanities, 9 (1975), 123-26.

16. Oakman, R. L. Computer Methods for Literary Research. Columbia: University of South Carolina Press, 1980.

17. Ross, D. and Rasche, R. H."EYEBALL: A Computer Program for Description of Style," Computers and the Humanities, 6 (1972), 213-21.

18. Ross, D. and Rasche, R. H. User's Instructions for EYEBALL. Rev. ed. Minneapolis: University of Minnesota English Department, 1976.

19. Spraycar, R. "CALAIS/Teach: The Lesson-Writing Component of a Computer-Assisted Language Arts Instruction System," 1982 ASEE Annual Conference Proceedings, 247-54.

20. Spraycar, R. "From CALAIS to DOVER: Using Computer-Assisted Programmed Instruction to Gather Data on Composition Deficiencies Automatically," Proceedings of the ASIS Annual Meeting, 19 (1982) (in press).

21. Van Tassel, D. The Compleat Computer.... Chicago and Palo Alto: Science Research Associates, 1976.

22. Zelazny, R. "For a Breath I Tarry," in Spinrad, N., ed., Modern Science Fiction, Garden City, NY: Anchor Doubleday, 1974.

The DISC Project

Shelley Yorke Rose
Carol Klenow
Instructional Computing
Oakland Schools
Pontiac, MI 48054

ABSTRACT
The DISC (Documentation and Integration of Software into the Classroom) Project, administered by the IICD (Interactive and Instructional Computing Department) of Oakland Schools in Pontiac, Michigan has produced the DISC Compendium, a collection of 91 software evaluations and documentation. The 565 page Compendium contains materials for the PET, APPLE, TRS-80, and Atari. Through DISC, 90 Oakland County educators were trained in evaluation techniques using an adapted MicroSift form and MECC documentation format. Combining these tools has provided teachers using the programs contained in the Compendium with a means to better implement the commercial programs to which they have access.

In addition to the Compendium, the DISC Project has produced a DISC Training Manual and Project Report which details each of the four workshops and includes presenters' outlines, hand-outs and transparency masters. Suggested session outlines and activities are include in the Training Manual.

The Disc Project was developed by the IICD during the 1981-1982 school year under a Title IV-C grant for $15,000. District superintendents and curriculum directors selected potential DISC participants using project criteria (computer literacy, ability to load and run a program, access to a classroom microcomputer, etc.) who, after an orientation session, attended four DISC workshops. DISC participants were trained in the use of MicroSift evaluation form and a format for the development of classroom viable support materials which was loosely based on MECC documentation. Their efforts in evaluating, pilot testing and creating support materials for selected commercial programs were rewarded with release time and the programs they evaluated.

To ensure the continuing contribution of the DISC Project to the area of commercial software evaluation and support materials development, the IICD is training additional Oakland County educators using the DISC Model. Resulting evaluations are being gathered and published in the IICD's newsletter "Remote Notes". Software publishers are being contacted with DISC results in the hopes that they will begin incorporating teacher needs into their documentation design.

COMPUTING FOR THE LEARNING DISABLED OR HANDICAPPED

Rita Horan
Warren R. Brown
Mary Russo
Dr. Nancy Jones
Sharon Smaldino
Partick Schloss

ABSTRACT: Using LOGO with Learning Disabled Students

Rita Horan, 10 Bayard La Apt 5, Princeton, NJ 08540

The Learning Disabled child has been typically described in the literature as having poor problem solving skills. Other descriptors of Learning Disabled children include such terms as: inpulsive, deficient in selective attention, an inactive learner, and users of deficient strategies in solving problems.

Much of the research on problem solving has focused on whether or not such children possess the underlying skills or abilities to solve problems. Studies have shown that learning disabled children have the ability to acquire and store information but lack spontaneous access to these processes. And furthermore they (Learning Disabled) seem to have an inability to generalize a previously learned problem solving strategy to a new problem.

Scientists and researchers in the data processing discipline have been studying the underlying processes of problem solving and refer to the process as information processing. Educators and researchers in the education disciplines address problem solving as a cognitive function which can be modified and trained to. Cognitive modification research has provided educators with procedures that have shown some durable and generalizable effects for Learning Disabled children. Specifically they have shown success with producing behavioral changes in hyperactive and impulsive children and children who have difficulties attending to school tasks.

During the 1970's a group of researchers at MIT developed a language called LOGO or Turtle Geometry. This language was specifically desgined to be something children could make sense of, to be something that would resonate with their sense of what is important: in order to learn something, first make sense of it. Turtle Geometry is learnable because it is syntonic. And, it is an aid to learning other things because it encourages the conscious, deliberate use of problem solving and mathematical strategies. The language is graphic oriented and users are encouraged to choose a project (problem) and through commonly used English commands (back, forward, left, right) create the steps which will solve the problem or complete the project. When the child hits a snag he/she is encouraged to use their own knowledge of the commands and previously learned information to fix the snag.

This study proposes to use LOGO as a tool to develop problem solving skills for Learning Disabled children. Tasks initiated either by student or teacher will be defined and each student will finish the task within a certain amount of time. Problem solving strategies based upon those used in cognitive modification will be used as well as LOGO.

This study will compare three groups. The subjects will be Learning Disabled students ramdomly assigned to the three groups. One group will receive LOGO instruction with a strategy based on cognitive modification techniques for problem solving. One group will receive LOGO instruction without problem solving strategies. The thrid group will receive neither LOGO nor problem solving strategies. Each group will be give a pre-test prior to instruction involving a problem solving task and a post-test problem solving task upon completion of instruction. A comparison will be made of the three groups based on pre-test and post-test scores to determine whether there is a difference between the groups' abilities to solve problems.

The independent variable will be the method of instruction or non-instruction. The dependent variable will be the post-test scores.

This study hopes to answer the following questions: Will Learning Disabled students who receive LOGO instruction with problem solving strategies differ from those who receive only LOGO instruction? Will students who receive LOGO instruction differ from those who do not receive LOGO instruction?

ABSTRACT: Project CAISH Second Year Update

Warren R. Brown, School Board of Sarasota County, 3450 Gogio Road, Sarasota,FL 33580

Project CAISH (Computer Assisted Instruction and Support for the Handicapped) is currently focusing on the needs of the mentally and emotionally handicapped population, ages 3-21. Continuing support is being provided for orthopedically handicapped students. The project uses systematic methods to provide microcomputer systems, software and courseware, and specialized hardware interfacing to address educational objectives of students that are specified by the teachers and specialists involved.

New interface devices will be demonstrated to simulate student interaction with the

microcomputers. New and modified software will be demonstrated, to include single switch control of LOGO graphics, drill and practice of Blissymbols, and other MECC software modifications.

A description of a 15 station microcomputer laboratory for mentally and emotionally handicapped students will be given. Security, accessibility, adaptive device needs, and software recommendations will be discussed. Anecdotal summaries of student uses of the microcomputer systems will also be presented.

One of Project CAISH's goals is dissemination of information. In this regard, both hardware and software experience can be useful to others in the educational community. A 46-page Interim Report will be available for distribution.

ABSTRACT: Project S.O.S.

Mary Russo, Project Coordinator, Dr. Nancy Jones, Media Specialist, The School Committee of Boston, 75 New Dudley Street, Boston, MA 02119

Project S.O.S. (Support for Occupational Students) at the Humphrey Occupational Resource Center uses computer-assisted instruction to help secondary level handicapped minority students develop the competencies in math, reading, language arts and problem solving needed for their occupational training programs. Using minicomputers donated by Digital Equipment Corporation and Apple microcomputers, students learn basic skills while developing the computer awareness needed in the increasingly computerized world of work. With the increasing use of microcomputers in business and industry for basic tasks, such as storage of typing in a word processing system or training programs for brake repair in the automotive industry, computer-assisted instuction is a bridge between basic skills development and occupational training. The content of thirty instructional software programs was aligned with a set of 104 basic skills competencies identified as necessary for the occupational training programs at the Center. Students take a computerized survey test and are scheduled for two 40 minute sessions per week based on their performanbce and the number of sessions required for task completion. The goal of the Project is the placement of 240 handicapped minority students into skills training jobs involving computer utilization.

Students are selected for the Project based on their scores on the Metropolitan Achievement Test: Reading and Mathematics, the Massachusetts Test of Basic Skills, referrals by instructors, and self-referrals. The course of instruction, designed to integrate the achievement of basic skills with occupational training, computer awareness and employability skills, is individualized through the use of student contracts. Further Project development includes the development of an item pool for computerized assessment specifically related to occupational training programs.

ABSTRACT: Relative Efect of Microcomputer Instruction and Teacher Directed Instruction on the Performance of Hearing Impaired and Normal Hearing Students

Sharon E. Smaldino, Patrick J. Schloss, Tri-County Special Education District, Dewey Building, 1000 N. Main, Anna, IL 62906

Scope and Purpose: With the increased interest in the use of microcomputers in the schools and with their value as an instructional tool still in infancy, the intent of this study was to raise the issue of time of instruction in its relation to performance of the student. A major assumption supporting the use of microcomputers is that teachers' instructional time is limited and the use of the computer may serve to augment and expand that instructional time. To date, this assumption has not been empirically validated. Comparing teacher direct instructional time with computer instructional time with the performance of the students as the dependent variable should provide educators with guidance for judicial use of microcomputers.

The unique learning and behavioral characteristics of hearing impaired students makes them particularly germane to the study of relative instructional efficiency. Traditionally, the teachers of the deaf have experienced the need to be highly redundant in their instruction of new material. The verbal nature of instruction makes teaching time lengthy and student performance a poor indication of comprehension. This study was to examine the efficiency of time of instruction by the teacher or the microcomputer. The performance of the deaf students compared with the time of instruction will support the instructional efficiency hypothesis.

Method: The subjects of the study consisted of students in the school who had minimal exposure to computer assisted instruction prior to the study. The subjects of the investigation ranged in age between 13 and 19. There were two categories of students, those with normal hearing and those with hearing loss averages in their better ear ranging from 60dB to 105dB. Six of the subjects in the hearing impaired group were male and six were female. The normal hearing group had no identified loss of hearing. Four of the subjects were female and eight were male.

The procedure was conducted over a five-day period with daily instructional segments of ten minutes in length. The computer program utilized for this instructional format was the unit "Hurkle" from the Elementary-Volume 1 diskette developed by the Minnesota Education Computing Consortium. An Apple II-plus microcomputer with a green phosphorous screen was the only equipment employed. The teacher direct instructional material coordinated with the material in the microcomputer program.

Each student was seen by the same teacher for a ten-minute instructional period. The groups of students were randomly split into those who would receive instruction directly from the teacher and those who only received instruction on the computer. A total of four groups were identified: the hearing impaired receiving teacher directed

instruction (HI-TD); hearing inpaired receiving microcomputer instruction (HI-M); normal hearing receiving teacher directed instruction (N-TD); and normal hearing receiving microcomputer instruction (N-M). A script was used by the teacher to keep the instructional presentation for all groups the same throughout all conditions.

Each student was given an opportunity to learn how to use the microcomputer by playing a hangman game, written by Wendell Bitter. Familiar spelling words were used. The students receiving microcomputer instruction were instructed that they would be learning new math material and to attend closely to the computer's instruction. Those students receiving the teacher directed instruction were told that to help them better understand material to be utilized on the computer, they would benefit from some instruction from the teachers.

Each instructional period was timed. Both the number of guesses and the total number of trials in each instructional segment were tabulated. A 2x2 ANOVA technique contrasting the rate of guesses and correct responses for the four identified groups will be conducted.

A GUIDE FOR THE PURCHASE OF A COMPUTER SYSTEM FOR A TWO-YEAR CAMPUS

by Laurena A. Burk

Department of Systems Analysis
Miami University-Hamilton, Hamilton, Ohio 45011

ABSTRACT

The increasing number of students in computer-intensive programs, together with a need for various levels of computing power in virtually all disciplines, has forced schools to allocate many thousands of dollars for the purchase of computing equipment. This paper suggests a strategy for selecting computing equipment appropriate to the campus' needs based on a data processing method known as IPO.

INTRODUCTION

Despite a general shortage of funds for education and declining enrollments in many college programs, there has recently been an increase in students in technical Associate Degree programs. Most of these programs require moderate to heavy use of computer equipment. At Miami University-Hamilton, for example, the Computer Technology program alone has grown 30% in the last year. Students must learn to program in four computer languages: Fortran, Assembly, Cobol, and PL/1. The increasing number of students in computer-intensive programs, together with a need for various levels of computing power in virtually all disciplines, has forced schools to allocate many thousands of dollars for the purchase of computing equipment.

Since two-year schools are frequently diverse in their degree and course offerings, any computing system that is to serve the entire campus must have extensive capabilities. At MUH, the computing needs are expanding as those teaching both technical and non-technical courses discover the computer as a helpful educational tool. As faculty recognize the value of the computer to their curricula, programming languages as well as word processing, graphics, and specialized software packages become a must. Yet, finding and purchasing affordable equipment that will serve faculty and students adequately, becoming an integral part of virtually every degree program, is no easy task. The following paper therefore suggests a strategy that will enable a campus to make an informed decision about purchasing computing equipment for its specific needs.

GETTING READY

A key person or group of people should be designated responsible for all or for specific parts of this decision making process. The total group, or project team, should be computer literate and have a vested interest in the utility of the computer system to be purchased. Otherwise, there is little hope that the campus will acquire the most practical and cost effective system. As the only full-time instructor in the computer technology program, I was the project manager and team at MUH by default. I managed the small but extremely busy computing facility, and my students were the heaviest users of the facility. Thus I became the campus computing resource person.

Since not all team members will have a technical background, this discussion is aimed at the intelligent but uninformed reader. While systems analysts have written many highly technical papers on computer procurement, these may be technically sophisticated for some members of the project team. A simple process-oriented approach such as IPO and a few chapters of a good data processing text may be sufficient to focus the involved educator's background. These texts are easily obtained through most libraries. A sample of appropriate texts is listed in the bibliography.

THE OVERALL PROCESS

The data processing method known as Input-Process-Output(IPO) is a useful technique for developing an orderly decision making strategy for a computer system purchase. Since an IPO chart provides a clear picture of what a program is to do, application programmers frequently find it helpful to prepare IPO charts when planning programming tasks.

Working out the logical steps in a program prior to actually writing and testing it on the computer saves time and prevents problems. If programming changes are required, the IPO charts provide the documentation necessary to describe the tasks performed by the program logically, enabling the programmer to update the program quickly and easily. An IPO chart divides the work into three basic funcions: INPUT, PROCESS, and OUTPUT. The INPUT section of the chart shows all of the data needed to run the program. The PROCESS section describes the manner in which all of the inputs will be manipulated. The OUTPUT section shows what the result of the process will be. This approach can be used in similar fashion to select a computer system.

First, begin gathering the facts governing the selection of a computer system and organize these into a useful format for study and comparison. The result will be a gradual distillation of information necessary for a practical decision. This collecting of facts is a continuous iterative process. The initial input will not provide enough information to make a final decision but will point out what additional facts must be gathered, studied, and organized before a final decision can be made. The form of an IPO chart is shown in Figure 1.

IPO CHART

PROGRAM:_____ PREPARED BY:_____
DATE:_____

INPUTS:	OUTPUTS:

PROCESS DESCRIPTION:

NOTES:

An example of an abbreviated IPO chart often used by programmers when planning a new program. It is changed as the input data, the output data, or the process changes.

FIGURE 1

COLLECTING FACTS

I recommend beginning to gather facts as soon as there is evidence new equipment will be needed in the near future. The following tasks should be completed before contacting potential vendors:

1. Compile a WISH-LIST by seeking input from all computer users, current and potential. A simple questionnaire can provide uniform facts. Put no cost restrictions on wishes.
2. Separate the list into two parts: a MUST-HAVE list and a NICE-TO-HAVE list.
3. Convert the WISH-LIST into a list of functions the computer system should be able to perform. Separate hardware and software functions.
4. Prioritize these functions.

A sample of questions is given in Table 1. These questions do not need to be extensive or statistically processed for a small school. Faculty are less likely to complete a lengthy questionnaire than a brief one. Additional faculty input can be sought as needed. If specialized needs are indicated, the faculty most closely associated with those needs should begin to research them. For example, if a faculty member foresees a need for computer-aided-design (CAD), he or she should begin to investigate available options in CAD computing and should ultimately specify the CAD functions.

Dividing the WISH-LIST into MUST-HAVE and NICE-TO-HAVE subcategories is necessary to avoid buying frills with no substance and to discourage buying glamorous but nonessential items. A digitizer tablet may appear to be essential to CAD functions, for example, but it is probably not essential for students and faculty who create drawings from scratch at a terminal. In fact, the digitizer would probably be used only to demonstrate how to enter an exising drawing into the computer via the digitizer. This does not mean necessarily that a digitizer is a frill but rather that it should have a lower priority than a plotter in a CAD application.

The MUST-HAVE and NICE-TO-HAVE categories should be subdivided to show hardware and software needs. Hardware is

SAMPLE OF QUESTIONS

1. Do you currently use the computer?
   a. If so, what are your projected needs over the next three years?
   b. If so, what are your projected interests over the next three years?
   c. How many students in your classes do you estimate would be involved in using the computer?
2. If you are not currently a user, do you plan to use a computer in the next three years?
   a. If so, what type of computing would you like to have access to?
   b. How many students in your classes do you anticipate would use the computer?

TABLE 1

MUST-HAVE LIST

HARDWARE FUNCTIONS

1. support 20 users
2. upgradeable to 60 on-line users
3. programmable ports
4. easy daily backup
5. output to large screen television
6. dial-in capability
7. graphics terminals
8. adequate disk storage
9. 2 color plotter
10. communicate with other computers for batch processing

SOFTWARE FUNCTIONS

1. User-friendly operating system
2. Compilers needed:
   Fortran
   a. Cobol
   b. PL/I
3. Ability to create graphs
4. Ability to program in interactive Basic
5. Financial software package

TABLE 2

the electronic and mechanical equipment needed to support the specified functions. Software is the set of programs which control the hardware and enable it to perform specified tasks. Separation of these hardware and software functions makes it possible to choose the best type of hardware configuration for the campus needs. Once the general hardware configuration has been chosen, vendors and appropriate software can be sought. A sample list of hardware and software functions, in order of priority, is shown in Tables 2 and 3.

AVAILABLE HARDWARE OPTIONS

Two major hardware options are available to schools purchasing computing equipment. The first is to develop a microcomputer lab with one or more brands of microcomputers. The second is to purchase a multi-user system that supports all the major functions and to plan on using microcomputers as remote workstations or for special purposes. In a school where computing supports non-technical programs, a microcomputer lab can be a viable hardware option. If a computer-intensive major is part of the curriculum, however, more computing power is usually needed. Thus, the hardware option chosen for MUH was the multi-user minicomputer system. Only minicomputer vendors were sought. If the major hardware option is not easily decided by the project team, a checklist can be used, similar to that found in Davis. *

* William S. Davis, Systems Analysis and Design A Structured Approach (Reading, Mass.: Addison Wesley, 1983),p 277.

CHOOSING VENDORS

The first four tasks of the decision making strategy are actually preparation for approaching potential vendors. Since fact gathering leads to meaningful organization and study of those facts, the organizing process produces functional specifications useful in choosing a vendor and in gathering important technical and cost information from that vendor. Comparing computer systems is easier and more accurate when specific and detailed information is at hand. The two most critical steps in the decision making process are selecting potential vendors and choosing the right one.

Determining the practicality of functions can be an overwhelming task. Priorities change when specific hardware, software, and price are considered. For example, a CAD-oriented system may be considered a MUST-HAVE. If projected CAD usage is significantly lower than demand for an application software-oriented system, however, the CAD priority may be infeasible as a MUST-HAVE. CAD equipment was projected to affect only one faculty member and twenty students at MUH, while twenty faculty, three degree granting programs and more than two hundred students per semester demanded software development capability. The CAD system had to be moved to a NICE-TO-HAVE priority because of high expense and lesser impact within the school. Yet this shift in priority was not agreed upon until after the vendors contacted had provided specific information . Potential vendors can provide essential facts that will enhance the final decision. Failure to identify appropriate vendors and to gather enough of these facts will hinder informed decision making.

At first glance, the number of vendors possibly meeting the campus' needs may be far too great to permit reasonable comparison. Although a total of no more than five vendors should be sought, it is preferable to limit your vendor choice to three. There are six vendor characteristics to look for before calling any sales representatives:

1. Limit your choices to vendors with a sales and support office within 100 miles.
2. Check vendor support record and reputation before contacting.
3. Verify each vendor's financial status.
4. Choose vendors which are technologically up-to-date.
5. Seek recommendations from other institutions.
6. Contact existing users' groups familiar with the equipment.

Once the vendors have been chosen, ask them to provide two general configurations. One should inlude all of the MUST-HAVES, the other should add the NICE-TO-HAVE hardware and software. Both configurations should include itemized pricing and monthly maintenance. Supplying vendors with a form makes the responses uniform and easier to evaluate later. A sample list of hardware and software items derived from the functional specifications is given in TABLE 4.

NICE-TO-HAVE LIST

HARDARE FUNCTIONS

1. Color graphics terminals
2. 8 color plotter
3. Digitizer
4. Additional disk storage
5. Communicate with other computers for interactive processing
6. Ability to transfer files from at least one brand of microcomputer.

SOFTWARE FUNCTIONS

1. CAD software
2. Database software
3. Debugging software
4. Electronic mail
5. CAI packages

TABLE 3

Additional data on each vendor can be obtained from users of that vendor's equipment. Request lists of sites with systems similar to the preferred configurations and visit these at the busiest possible times. Ask the operators and system managers about performance, vendor support, ease of use, and response time. Note any problems mentioned and any outstanding features, either positive or negative.

HARDWARE ITEM	PURCHASE PRICE	MONTHLY MAINTAINENCE
1. Central processor		
2. Communication to IBM 370		
3. Tape drive (45 ips)		
4. Disk drive (200 megabyte minimum)		
5. Terminal (with printer port and advance video)		
6. graphics terminal		
7. line printer (300 lpm)		
8. battery backup		
9. modem (1200 baud)		
10. cable for 20 terminals		
SOFTWARE ITEM		
1. Basic		
2. Cobol		
3. PL/1		
4. Software for communication to IBM 370		
5. Financial software package		

TABLE 4
Preliminary form for vendors to complete prior to bidding

Upgrade costs should be investigated, as well as purchase and monthly maintenance costs, since upgrading will inevitably be necessary at some future time. Ask prospective vendors to include projected upgrade costs on their information forms. Knowing the cost per additional workstation, including cables, ports, and memory, is helpful in estimating future costs and determining if the proposed configuration is adequate. If upgrade costs are too high, the proposed system is probably too small. NEVER consider a system that cannot be upgraded.

Non-biased technical information can be obtained from Datapro reports. Datapro publishes objective technical summaries of all classes of computer systems and polls users of the equipment for their opinions. * These user summaries are particularly noteworthy in ease of operation, maintenance service, and documentation. Documentation is the vendor provided set of manuals which give the user the directions needed to use the system components appropriate to the task. Quality documentation is extremely important in an academic environment because of the variety of levels of faculty and student experience. Without it the system will be greatly under-utilized.

MEASURES OF EFFECTIVENESS: THE FINAL DECISION

After the input from users and vendors has been collected, the project team will has several alternative computer systems to choose from. Occasionally, one feature of a computer system may make it so far superior for a particular campus' purposes that it is the apparent choice. Ordinarily, however, the key people responsible for organizing and studying will find additional prioritizing decisions necessary before they can decide on a vendor. Choosing between vendors, when two or three appear to be able to supply appropriate hardware and software within budget constraints, requires further careful measurement of system capabilities. The next step in the decision making process, therefore, is to analyze these alternatives, using technical, economic and operational criteria. These criteria may be summarized as follows:

1. Technical criteria
   a. Functionality
   b. Level of technology of the hardware and software
   c. Richness of available software
   d. Security
2. Economic criteria
   a. Purchase or lease cost
   b. Maintenance contract costs
   c. Upgrade costs

* Datapro Reports. Delran, New Jersey: Datapro Research Corporation.

3. Operational criteria

   a. Vendcr support

   b. Security

   c. Implementation schedule

The first technical measure is the vendor's response to the question, "How well does the system perform the needed functions?". For example, two systems may supply a Fortran compiler, but one is Fortran '66 and the other is Fortran '80. Thus the second system performs that function better.

The level of technology of equipment is another important consideration. The quality of education an institution provides is reflected in how up to date its equipment is. Better computing equipment attracts more and better students and their skills are more marketable upon graduation if learned on state of the art equipment. In addition, newer computing equipment is more reliable and cheaper to maintain. New equipment becomes obsolete quickly enough; saving a few dollars on older technology is penny wise and pound foolish.

The richness of software available on the market should be considered at the time of purchase. As more faculty and staff learn to use the computing resources at hand, and as the variety of applications software increases, additional software purchases will become cost effective investments. Our new computer system has not yet been delivered; yet those who expressed special needs are already searching for additional software.

Cost is the first economic measure. The number and quality of functions performed within budget limitations must be considered, as well as additional costs such as maintenance and upgrade. Further cost-related consideration is the availability of compatible equipment. Frequently, compatible equipment such as terminals or printers, can be obtained more cheaply through a distributor. This can be factored into cost of upgrade.

Smooth daily operation of a computer system depends heavily on vendor support. The vendor's support record, both technical and maintenance, is a good indication of how much the system will be used. If technical problems can be ironed out easily, minimizing machine repair time, more people will use the machine. Software updates are a must and should be included in software maintenance since a system becomes unneccessarily obsolete without them. Documentation too falls under technical support. Verify that all manuals are readable.

Although the type of computer system security needed in an academic environment should be readily available on most multi-user systems, the question of adequate security merits careful scrutiny. Passwords and the ability to prevent students from accessing each others programs are absolute minimums. Be sure the proposed system offers appropriate security. Examining security procedures within another school environment may give additional insight into methods your institution can adopt.

The third operational criteria is the implementation schedule. The neccesary equipment should be installed in sufficient time to give faculty members adequate preparation for classroom use. Those faculty already familiar with computers will need one semester to adapt to a new computing environment. Introducing unfamiliar faculty to the computer as an educational tool will follow. If curriculum objectives are affected by the availability of a computer system, then the implementation schedule should be a criteria in evaluating the vendor proposals.

Once it is apparent that two or three of the vendors meet the campus' needs and measure up fairly well, request formal bids. Although this may delay delivery by a few weeks, it will most assuredly reduce the price or bring a few of the unaffordables within reach. When requesting these bids be sure to demand a complete installed system price. This includes delivery, installation, and bringing the system on-line . At this point, price can be the final deciding factor. Let the vendors know that it is not the only criteria for final selection. The results of the bidding process can be surprising. At MUH, the three vendors that were asked to bid were IBM, Hewlett-Packard, and Digital Equipment Corporation (DEC). Only Hewlett-Packard and DEC submitted bids. Both vendors measured up well, DEC's richness of software and newer technology winning the purchase contract. Hewlett-Packard is currently offering free software to educational institutions, which makes them extremely competitive.

The decision making process does not end when the equipment is purchased and installed. Instead, as long as computing is seen as an integral part of the educational process, the effectiveness of the system must be maintained through

continuous reassessment. This means project teams keep abreast of changes in campus' needs and new developments in the industry which can better meet those needs. Here again, the IPO process facilitates efficient decision making, building on existing knowlege to achieve consistent effectiveness.

BIBLIOGRAPHY

Davis, William S. Systems Analysis and Design. Reading, Mass.: Addison Wesley Publishing Co., 1983. pp. 274-279

Davis, William S. Operating Systems A Systematic View. Reading, Mass.: Addison Wesley Publishing Co., 1983.

Fitzgerald, Jerry; Fitzgerald, Ardra; Stallings, Warren. Fundamentals of Systems Analysis. New York: John Wiley and Sons, 1981. chapter 7

Weinberg, Victor. Structured Analysis. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1980.

EXTENSIVE COMPUTER GRADING OF ID-INDIVIDUALIZED HOMEWORK PROBLEMS

M. J. Maron

Department of Applied Mathematics and Computer Science
University of Louisville, KY 40292*

Abstract

This paper describes an efficient methodology in which a computer is used to grade ID-individualized homework problems on demand while keeping records for these homeworks. The methodology was developed at the University of Louisville and used there with undergraduate courses in statics and numerical analysis. Results so far indicate that it can be an effective pedagogical tool for any course in which homeworks have numerical answers.

Rationale

The following feature distinguishes this methodology from most existing schemes for computerized grading of individualized problems[1]:

> The student does not have to go to the computer to get individualized problems. Instead, homework assignments are given in the "traditional" way, that is, on a single duplicated sheet which is distributed to all students in the class (Figure 1). The individualization is imposed by the following simple device: Problems are given in terms of A, B, C, D, E, and F, where ABCDEF denotes a 6-digit student ID. For example, a problem may require the student to evaluate
>
> $$(3.AB)(5.CD + 12.F - E)$$
>
> A student whose ID is 123597 will calculate
>
> $$(3.12)(5.35 + 12.7 - 9) = 28.236$$
>
> Any other student, having a different ID, should get a different answer. In this way, all students solve the same problem but no two students should get the same answer.

The 6-digit IDs are stored when the student files are created. Once this is done, there is no further need to store or generate individualized randomized numbers for each student for each homework.

The problems are solved in the "traditional" way, i.e., with pencil, paper, calculator, etc., at a desk, not at the computer terminal. It is only after solutions are obtained that the student uses the grading program. A grading session consists of identification, submitting answers to some or all problems of a single assignment, grading, and getting a summary (Figure 2). It takes about 3 to 6 minutes to submit and grade 16 to 18 answers. The program does the grading by accessing the file of student IDs and checking whether the submitted answers agree with the "correct" ones calculated by an appropriate subroutine of the program to within a prescribed accuracy.

The student has a fixed time period (e.g. 10 days) to get the best possible mark on a given assignment. During that time he or she may consult with the instructor and/or classmates to correct and resubmit those answers that were wrong. Such a time limit effectively eliminates the problem of procrastination that generally plagues "self-paced" schemes[2], while still giving the student a chance to achieve proficiency on each segment of the course.

The methodology described above ensures the following:

Each student does his or her own work; but students may consult with each other about how to get correct answers. (This type of collusion is considered as desirable.) Simply copying someone else's work (considered undesirable) does the student no good whatsoever.

The grading is uniformly fair and immediate. Rather than wait for a human grader, the student immediately knows which answers are wrong and can begin to correct them at once. Answers can be re-submitted repeatedly if necessary until either the student is satisfied with the grade on that assignment or the time limit for that assignment has expired.

The instructor can assign as much routine drill as deemed necessary without having to face the tedium of grading it and recording the results. Instead, human grading energy can be expended where it is needed, namely on projects and assignments which require judgment, analysis, comparison, etc.

There is no major crisis when the computer goes down because the work is done away from the terminal. If necessary, a simple adjustment of the

*This material is based upon work supported by the National Science Foundation under LOCI Grant No. 79-00864.

COURSE # | 207 | ASSIGNMENT # | 2 | DATE ASSIGNED | January 24,1983 + 14 |

sec 3

CLASSLIST # | ____ | ID = | A | B | C | D | E | F |

REMINDERS: 1. Substitute digits of your ID for ABCDEF.

2. Unless otherwise indicated, use your calculator accuracy in all intermediate steps, storing intermediate values if possible.

3. Submit answers (A1, A2,...) to at least | 5 | significant digits.

	Answers	Points
Q1. Let $f(x) = x^3 - (50.E)x = x^3 - (50.\_\_)x$.		
If the Newton Raphson (NR) method is used	A1 =	6
with $x_0 = 4.2CD = 4.2\_\_\_\_$, then	A2 =	6
x_1 = | A1 |, x_2 = | A2 |, x_3 = | A3 |	A3 =	6
whereas if $x_0 = 3.FB = 3.\_\_\_\_$, then	A4 =	6
x_1 = | A4 |, x_2 = | A5 |, x_3 = | A6 |	A5 =	6
	A6 =	6
Q2. Let $f(x) = x^3 - (50.E)x = x^3 - (50.\_\_)x$.		
If the Secant (SEC) Method is used with	A7 =	6
$x_{-1} = 4.3DE = 4.3\_\_\_\_$ and $x_0 = 4.2CD = 4.2\_\_\_\_$,	A8 =	6
x_1 = | A7 |, x_2 = | A8 |, x_3 = | A9 |	A9 =	6
whereas if it is used with $x_{-1} = 2.9E = 2.9\_\_$	A10=	6
and $x_0 = 3.FB = 3.\_\_\_\_$,	A11=	6
x_1 = | A10 |, x_2 = | A11 |, x_3 = | A12 |.	A12=	6
Q3. Let $x_0 = 3.D = 3.\_\_$, $x_1 = 1.C = 1.\_\_$, and		
$x_2 = 0.4F = 0.4\_\_$.		
If the convergence is exactly linear, then	A13=	6
$\Delta x_k/\Delta x_{k-1} = C_1$ = | A13 | and x_3 = | A14 |	A14=	8
If the convergence is exactly quadratic, then	A15=	6
$\Delta x_k/(\Delta x_{k-1})^2 = C_2$ = | A15 | and x_3 = | A16 |	A16=	8

Figure 1: Sample Computer Graded Assignment

```
HI THERE.  PLEASE ENTER SECTION #, HOMEWORK #:  1, 1

ENTER CLASSLIST NUMBER, ID:  1, 123456

ENTER LAST NAME:  MARON

DO YOU WANT TO 1) SUBMIT ANSWERS OR 2) GET SUMMARY?
1 OR 2?  1

REMINDER:  HIT <RETURN> KEY TO SKIP A QUESTION
           CTRL/U TO RE-ENTER AN ANSWER
           CTRL/Z TO DISCONTINUE "A=" PROMPTS.

 A1=   OK
 A2=   JK
 A3=   OK
 A4=   OK
 A5=   OK
 A6=   OK
 A7=   OK
 A8=   OK
 A9=   -14.9

A10=   OK
A11=   .4125

A12=   OK
A13=   OK
A14=   OK
A15=   5.24448

A16=   8.26305

A17=   37.2045

PLEASE WAIT WHILE GRADING:

 A9=  -14.9000
A11=  0.412500
A15=   5.24448
A16=   8.26305
A17=   37.2045

CORRECT ANSWERS:

  A9
 A11
 A15
 A16
 A17


SUMMARY FOR HOMEWORK # 1

  A#          1     2     3     4     5     6     7     8     9
POINTS:       5     5     5     5     5     5     7     7     7

  A#         10    11    12    13    14    15    16    17
POINTS:       7     6     6     6     6     6     6     6

TOTAL POINTS: 100  OUT OF A POSSIBLE 100
WELL DONE!!!

ENTER ANY COMMENTS OR COMPLAINTS (THEN HIT<RETURN>).
 IF NONE, JUST HIT <RETURN>

?  THANKS, I NEEDED THAT

Y'ALL COME BACK SOON.
```

NOTE: UNDERLINED CHARACTERS WERE ENTERED AT A TERMINAL.

Figure 2: Illustrative session with grading program

program files can extend the grading period for selected homeworks.

Applicability

Homework problems and the associated subroutines for grading them have been written for a freshman level course in statics and for a junior level course in numerical analysis. However, the grading program can be used with any course for which there are routine problems which have numerical answers. This attribute characterizes most "core courses" in an engineering or natural science curriculum, and most quantitative courses in a social science or business curriculum.

Desirable Operational Features

Aside from the general benefits described in the Rationale, the use of the computer grader in several courses simultaneously offers the following operational benefits:

Student files for each course are easy to set up. The information required for a student record (viz. name and an ID) is precisely the information provided by the registrar on alphabetical classlists. So this classlist information can be dumped onto a temporary file which can then be read into the appropriate fields of the grading program files by a simple utility. Only minor modifications (late registrants, etc.) need be keyed in manually.

The storage requirements are minimal. A single grading program can serve several courses. Since there are no randomized numbers for the computer to generate or store for each homework, the program and files are quite small. Object code for the program requires 65 blocks (about 40K) of disk storage on a DEC 1090; the grading subroutines are generally short and should add no more than 10-25 blocks per 3 semester hour course (Figure 3). With the use of bit packing, all necessary information for one student's performance on one assignment is stored in a single word (which can be as small as 32 bits). As a result, the file for a class of 35 students who are given 25 computer-graded assignments requires only 20 blocks (about 12K) of disk storage on a DEC-1090.

Teachers of different sections of a particular course can assign homework independently. This flexibility was written into the grading program to ensure that instructors using it do not feel that someone else's "gimmick" is being forced upon them. Rather, each instructor can teach the course in the same order and with the same emphasis that he or she would use without the computer grader. This flexibility also makes it easy for an instructor to modify the syllabus from semester to semester if desired.

Some of the time which student assistants currently spend as graders can be spent programming grading subroutines. These subroutines can be written without any knowledge of the design of the grading program. Indeed, a student with reasonable programming expertise can write a subroutine for a new problem in 20-40 minutes by merely mimicking Figure 3. This is more challenging, more interesting, and certainly no less of a learning experience than grading the same problem(s) repeatedly for all students in one or more classes.

Once the grading subroutines are written, a teacher can choose to re-use assignments of previous semesters without fear of students copying the homework of the previous semester.

Results

The grading program was used intensively with three sections of ESC 205, a freshman-level statics course, and two sections of AMCS 207, a junior-level numerical analysis course in the Spring 1983 semester. All sections had an initial enrollment of about 35. Each week, two short computer-graded homeworks (typically 1-8 answers) were assigned in ESC 205 and one longer computer graded assignment (typically 12-18 answers) was assigned in AMCS 207. The grading period for these assignments varied between 7 and 14 days. Although the final results are not yet available, some results are evident.

Pedagogical Effectiveness

After eight weeks, 89 students in the three sections of ESC 205 and 68 students in the two sections of AMCS 207 were still using the computer grader. The grades they received on the first 15 homeworks in ESC 205 and the first 6 homeworks in AMCS 207 are tallied in Table 1.

Grade Range	Number in ESC 205	Number in AMCS 207
=100%	908 (68%)	351 (86%)
90-99%	16 (1%)	13 (3%)
80-89%	37 (3%)	15 (4%)
70-79%	47 (4%)	10 (2%)
60-69%	41 (3%)	7 (2%)
up to 59%	286 (21%)	12 (3%)

Table 1

Evidently, most students were motivated to get the highest possible grade on each assignment. This is further supported by the fact that students used an average of about 4 to 8 grading sessions to get these grades, and some students used more than 15 sessions for some homeworks!

This high degree of motivation resulted in many students coming to my office when they (and perhaps their friends as well) ran out of ideas about how to do problems I assigned in AMCS 207. Aside from the obvious benefits for the students, these encounters pointed out several poorly conceived problems and some shortcomings in the grading subroutines.

Results of student questionnaires during an earlier trial with only AMCS 207 indicated that (a) the students enjoyed and believed they

```
      SUBROUTINE HW2
      COMMON /HWK/ A, B, C, D, E, F, IPROB, ANS(9)
C
      GO TO (1, 1, 2), IPROB
C
C     ****     Q1 AND Q2     ****
1     C1 = 50. + E/10.
      X0 = 4.2 +.01*C +.001*D
      X00 = 4.3 +.01*D +.001*E
      IF (IPROB .EQ. 1) X00 = X0
      CALL ROOT(IPROB, X00, X0, 3, C1, ANS, 0)
      X0 = 3. + .1*F + .01*B
      X00 = 2.9 + .01*E
      IF (IPROB .EQ. 1) X00 = X0
      CALL ROOT(IPROB, X00, X0, 3, C1, ANS, 3)
      RETURN
C
C     ****        Q3         ****
2     X0 = 3. + D/10.
      X1 = 1. + C/10.
      X2 = .4 + F/100.
      DX0 = X1 - X0
      DX1 = X2 - X1
      ANS(1) = DX1/DX0
      ANS(2) = X2 + ANS(1)*DX1
      ANS(3) = ANS(1)/DX0
      ANS(4) = X2 + ANS(3)*DX1**2
      RETURN
      END
C
      SUBROUTINE ROOT(M, XPREV, X, ITER, C1, ANS, IBUMP)
      DIMENSION ANS(9)
      FUNC(X) = (X*X - C1)*X
      DERIV(X) = 3.*X*X - C1

      FPREV = FUNC(XPREV)
      DO 50 I=1,ITER
         F = FUNC(X)
         IF (M .EQ. 1) SLOPE = DERIV(X)
         IF (M .EQ. 2) SLOPE = (FPREV - F)/(XPREV - X)
         XPREV = X
         X = XPREV - F/SLOPE
         FPREV = F
         ANS(I+IBUMP) = X
50    CONTINUE
      RETURN
      END
```

Figure 3: Subroutine to grade assignment in Figure 1

learned from the computer assignments, and (b) their attitude toward computers was more positive after the course, perhaps due at least in part to the computer grading[3].

Time Involved

The terminals used were connected to a DEC-1090 via 300 baud lines. The average connect time over all sections seems to be around 18 minutes (total) per student for one assignment, or about 3 minutes per grading session. Most likely, initial sessions took longer and later "fixup" sessions were shorter. Grading sessions can be made somewhat shorter by using higher speed lines.

Another factor that will reduce the total grading time is the removal or correction of problems that were poorly worded, ill-conceived, or improperly graded. (All of the ESC 205 problems and about half the AMCS 207 problems were written about a week before they were assigned.) These caused many unnecessary grading sessions.

Cost Involved

At a university that already has remote terminals available to a host computer and makes no real dollar charge for instructional computer use, the only possible major expense is for the time spent preparing and debugging the subroutines for grading problems for a particular course. Since problems can be re-used effectively, and subroutines to grade new problems can be written by student assistants, this charge should be reasonable. Even if disk storage and connect time are billed to a real dollar account, the cost should compare favorably to the cost of paying a student the minimum wage to grade papers (with not nearly the effectiveness of the computer grader).

Summary and Conclusions

This paper described an ID-individualized methodology in which homework grading, but not learning, is done interactively at a terminal. The results so far indicate a high degree of both pedagogical and cost effectiveness. Although more extensive study is needed, these results suggest that for courses in which there are routine problems having numerical answers, this methodology may prove to be superior to both "traditional" hand-graded schemes and more extensive CAI schemes.

Bibliography

1. "Computer Aided Assignments in Electrical Engineering Education", W. J. Smolinski, ASEE Educational Research Methods Journal, Vol. 10, No. 3, (Spring 1978).

2. "Does PSI Work Because it is Self-Paced?", M. J. Maron and L. D. Tyler, Proceedings of 1976 Frontiers in Education Conference, Tucson, AZ.

3. "The Effects of Computerized Grading on Student Attitudes Toward Computers", M. J. Maron (To Appear).

AUTOMATIC SYLLABUS GENERATOR (ASG)

Asad Khailany, Marc B. Schubiner, A.M. VanderMolen

Department of Operations Research and Information Systems
Eastern Michigan University, Ypsilanti, Michigan

ABSTRACT

Many instructors are facing problems in preparing syllabi for multi-section courses. Often these are core courses, such as introduction to computer science, information systems, mathematics, and statistics. It is difficult for the various instructors to be consistent in teaching required course material established by the administration. Frequently, individual instructors prepare their own syllabus for a section, leading to many different syllabi for the same course. Furthermore, typing the syllabi is an added burden to the departmental secretaries. The Automatic Syllabus Generator (ASG) is a menu driven software package written to generate any type of syllabus. Such a system has a number of economical and administrative advantages. However, it may raise some questions with respect to academic freedom. This paper discusses the design, implementation, advantages and disadvantages of ASG.

INTRODUCTION

In the last decade considerable emphasis has been placed on the content and quality of syllabi. Usually, the administration requires instructors to prepare their syllabi in accordance to the institutions' guidelines which may contain all or some of the items in figure 1. Further, the administration expects that instructors hand out their syllabi to the students at the beginning of the semester. In many institutions, the syllabus is considered to be a contract between the instructor and the students. It is used as the base to settle any dispute between the instructor and students during or after the semester period. Therefore, a syllabus is an important document not only for the student but for the instructor as well.

The situation often arises in large educational institutions where a course is taught in several sections by different instructors. This situation lends itself to several problems. Among these problems are that of consistency of material taught in each of these sections and in subsequent semesters. Also there is an increased work load on the secretarial staff in preparation of several different syllabi. The difficulty that new or part time instructors have in determining exactly what should be taught in the course is an additional problem. Another problem is that with varying syllabi content, students may not meet their objectives in taking the course. Finally, problems can arise with meeting accreditation standards and student preparation for subsequent courses.

The system can have its drawbacks. The most striking, in the case of the unified syllabus, is that of academic freedom. The question arises, how much can be dictated to the instructor as to how the class will be taught and the exact subject matter covered. This is a fine line and could possibly destroy a course whose reputation was built on the personality of the instructor. Furthermore, if the administration does not update the required documents, the syllabus may not keep up with new innovations and future technological changes. It is possible that course material could become stagnant or dated.

A solution to some of these problems is presented here. A computer-aided Automatic Syllabus Generator (ASG) was written to assist the instructor in creating a course syllabus. Additionally, to insure standardization of the syllabus, certain aspects are fixed such as institutional policies and course goals. Other requirements are fixed as to their inclusion in the syllabus, however, their content may vary according to the individual instructor.

The general outline of the syllabus develops with the appropriate body (faculty, administration, etc.) drafting the objectives of the course and other required documentation. These results are placed in files which are used by the ASG to automatically generate a syllabus. Prior to creating a syllabus, the individual instructor would include their personal criteria and information. This includes office hours, grading system, project, text book, etc. The resulting syllabi can then be printed en masse by the computer. Also the syllabus can be stored on the computer and students could print their own copy. In both cases, considerable secretarial time and expense can be saved.

Figure 1 shows the list of parameters for the Automatic Syllabus Generator (ASG). Those denoted as fixed are parameters which are consistent throughout all syllabi for a course. Those denoted as required are parameters which must be included in the syllabus but the content may vary from instructor to instructor.

SYLLABUS OUTLINE

Name of the course
Semester
Name of the professor
Office room and hours
Telephone
Meeting time and place
* Goals and objectives
* Text book
* References
* General outline of the course
Detailed course outline
Grading system
Homework/Term project
Policies:
- * Grading homework projects
- * Absenteeism of exams
- * Make up exams
- Class contribution
- Class absenteeism
- * Late homework
- * Cheating
- * Granting Incomplete

Figure 1: Items recommended to be included in the syllabus

* – denotes the fixed items

SYSTEM DESCRIPTION THROUGH THE DATA FLOW DIAGRAM

The Automatic Syllabus Generator (ASG) is composed of five subsystems, see figure 2. These are:

1.1 – The Initial Process
1.2 – Create Processes
1.3 – Update-Time-Bound-Information
1.4 – Update Syllabus-Stable Information
1.5 – Generate Syllabus

The Initial Process Subsystem (IPS), see figure 3, has two components: Enter and Verify subsystems. The IPS accepts the course-information data flow, which is comprised of course prefix (such as MAT, ORI, CCS, etc.), course number, course title and credit hours, from the user. Before further processing of the data flow, the IPS displays it back to the user for verification. After confirmation from the user, the IPS passes the course-information data flow to the Create Process Subsystem (CPS).

The components of the CPS are: Input, Verify and Write subsystems. The Input subsystem accepts the time-bound-information data flow from the user, and calls the Verify subsystem to confirm the user's input. Next, the CPS calls the Write subsystem to write the time-bound-information, which consists of the semester (such as fall, winter, spring or summer), year, instructor's academic title, name, office location, telephone number, office hours, course section ID, section number, class room number, building and meeting time in the SSYL file.

The Update-Time-Bound-Information Subsystem utilizes three subsystems: Read, Update and Verify subsystems. The Update-Time-Bound-Information Subsystem is used by the instructor to modify an already existing syllabus or to modify the time-bound-information data flow.

The fourth subsystem, Update-Syllabus-Stable-Information Subsystem (USSIS), utilizes three subsystems: Read, Edit, and Write subsystems. This subsystem is used to update the stable-information files.

The last subsystem of ASG, Generate Syllabus Subsystem (GSS), utilizes two subsystems: Write Syllabus and Print Syllabus subsystems. This subsystem retrieves time-bound-information data flow from the SSYL file and stable-information data flow from the stable files and generates the actual syllabus by writing it in the file SYLLABUS. Further, the GSS prints the desired number of copies of the syllabus. The source of the stable-information data flow is the stable files.

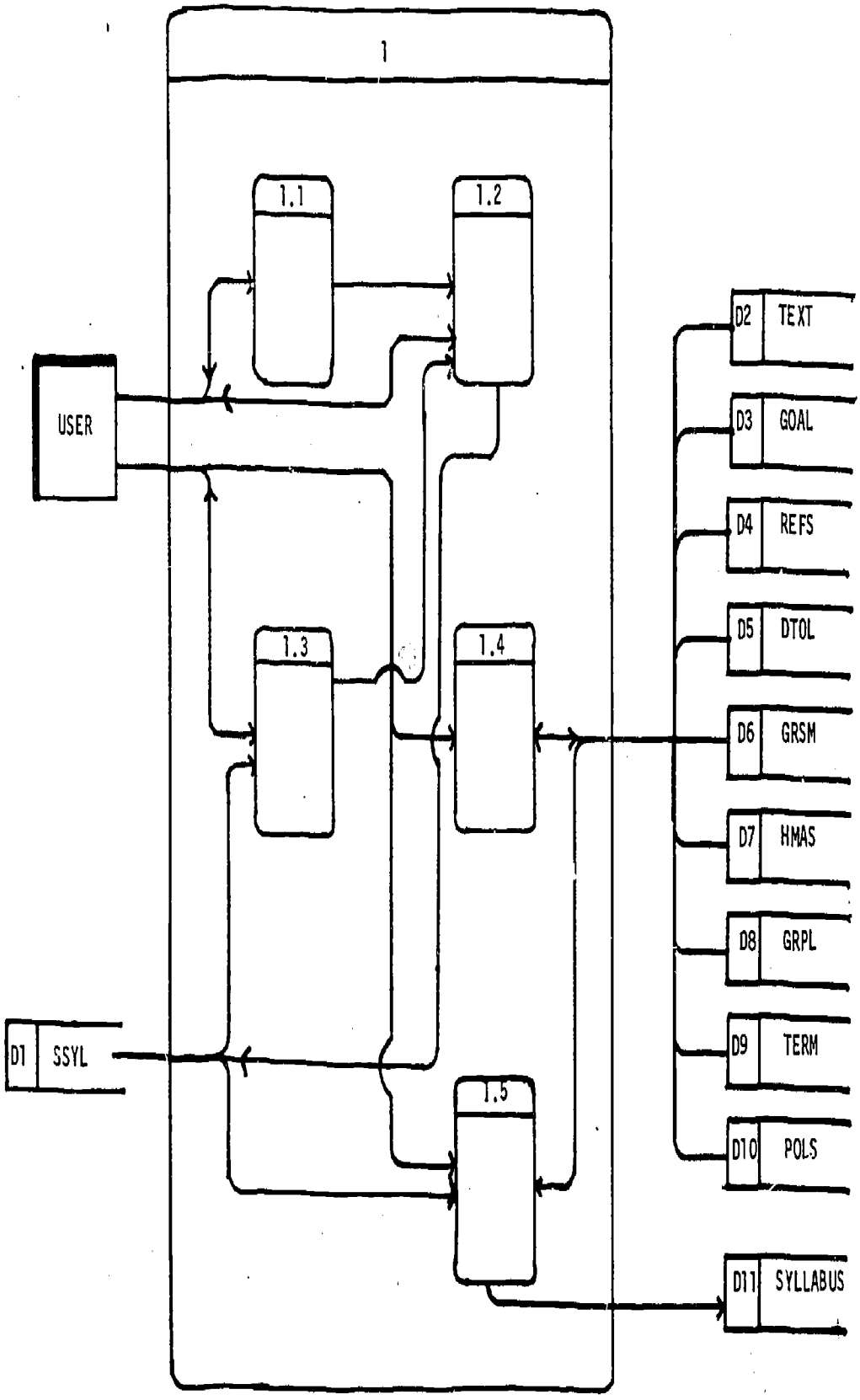
A copy of a syllabus produced by the ASG is in the appendix.

The following nine files make up the stable files:

TEXT file.	This file contains the required and optional text books.
GOAL file.	This file contains the general and specific goals and objectives of the course.
REFS file.	This file contains all recommended references for the course.
DTOL file.	This file contains detail course outline.
GRSM file.	This file contains grading system and grading scale.
HMAS file.	This file contains the required term projects and homework assignments and their due dates.
GRPL file.	This file contains homework and term project grading policies.
TERM file.	This file contains the title and detail description of the required term paper and projects.
POLS file.	This file contains policies regarding absenteeism from exams, make up exams, cheating, granting incomplete and class absenteeism.

IMPLEMENTATION

The Automatic Syllabus Generator (ASG) is written in Fortran-10, Version 6, and is implemented on a DECsystem-10 with a KL10B processor. The code follows the ANSI 1966 Fortran standards permitting ease of transfer to different processors. The system is totally modularized and is comprised of 55 modules. On the DECsystem-10, each block is 128 words or 640 characters and the processor has 36 bits. The ASG source code requires 120K or 186 blocks of storage on the DECsystem-10. The ASG in relocatable form uses 90K or 140 blocks and the execution form occupies 50K or 80 blocks. The compilation time is approximately 10 seconds.



1. Automatic Syllabus Generator

1.1 Initial Process

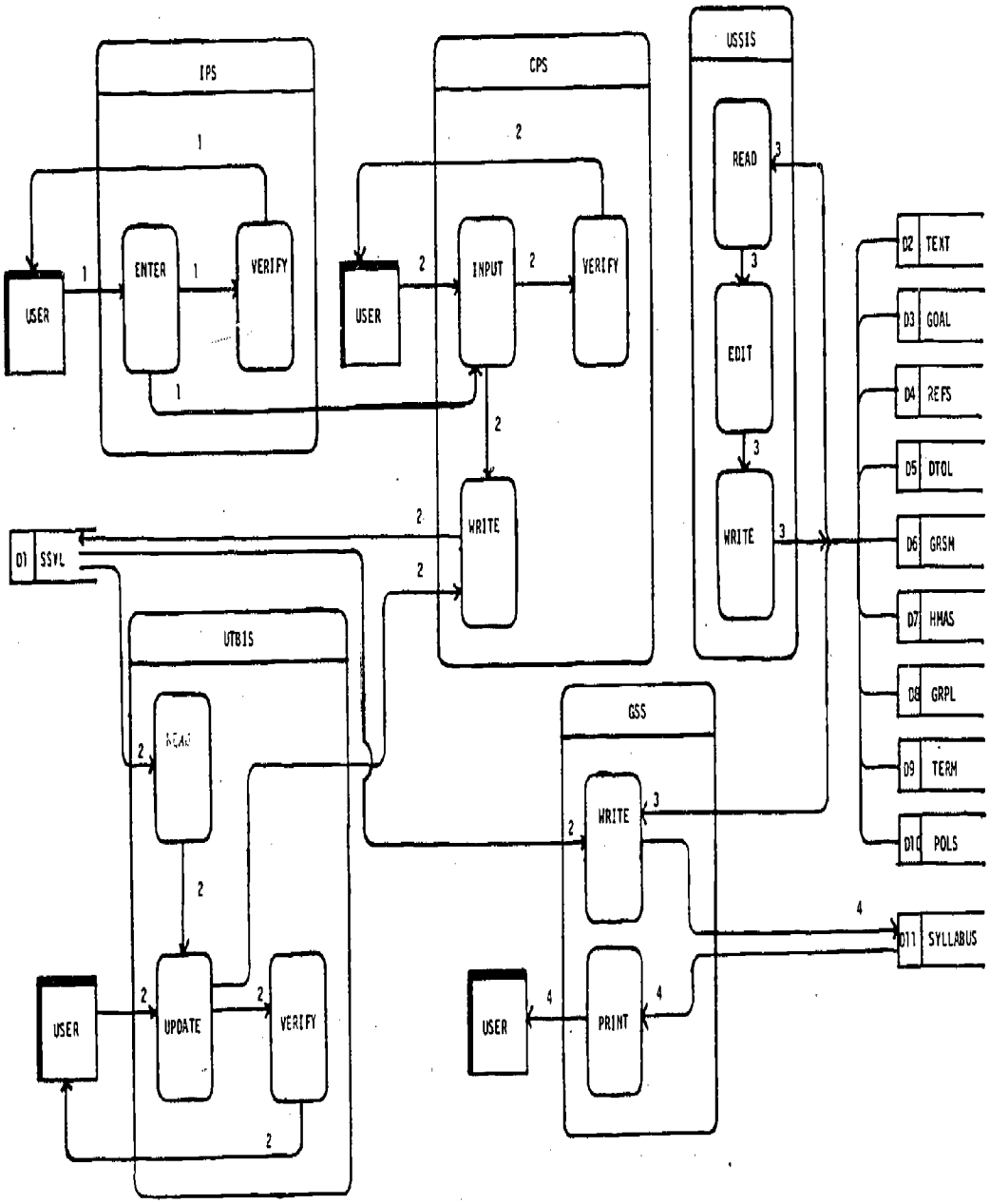
1.2 Create Process

1.3 Update-Time-Bound-Information

1.4 Update-Syllabus-Stable-Information

1.5 Generate Syllabus

Figure 2: The data flow diagram of the ASG.



1 -- Course Information

A -- Course Prefix

B -- Course Number

C -- Course Title

D -- Credit Hours

2 -- Time Bound Information

A -- Semester

B -- Year

C -- Title

D -- Name

E -- Office Room

F -- Building

G -- Telephone

H -- Office Hours

I -- Hours By Appointment

J -- Section ID

K -- Section Number

L -- Class Room

M -- Building

N -- Meeting Time

3 -- Stable Information

4 -- Syllabus Information

Figure 3: The detailed data flow diagram of the ASC.

A BRIEF DESCRIPTION FOR THE USER

The user begins the Automatic Syllabus Generator(ASG) process by entering the course-information. The course-information consists of the course prefix, course number, course title and credit hours. Next the user has the choice of:

1 – creating a new syllabus,
2 – updating the time-bound-information (TBI),
3 – updating the syllabus-stable-information (SSI),
4 – writing the syllabus on the disk and printing it on the line printer,
5 – print multiple copies of a syllabus.

The TBI consists of the file SSYL, the file has the following elements: semester, year, academic title and name, office room number and building, telephone number, office hours, hours by appointment, section ID and number, class room number and building, and meeting time. The SSI consists of the following files: TEXT, GOAL, REFS, DTOL, GRSM, HMAS, GRPL, TERM,and POLS. The syllabus consists of the file containing the TBI, and all the files containing the SSI.

Creating a new syllabus option is chosen if the user has never generated a syllabus for the course. The ASG stores the TBI, input by the user, into the SSYL file. Next, the ASG generates a copy of the syllabus on the disk and the lineprinter. This syllabus is considered a temporary copy for the users inspection. After receiving verification from the user, the ASG generates a permanent copy.

Option 2, updating the time-bound-information option, is chosen if the user wants to update an existing syllabus and needs to change the TBI. The user can update each element of the SSYL file individually with the new value written into the SSYL file. The ASG generates a new temporary copy of the syllabus if no update is required in the SSI files. However, the user must continue on to the next option if SSI requires updating.

Updating the syllabus-stable-information option is chosen if the user needs to update the SSI in the SSI files. The user can edit each file separately and when updating is completed a new temporary copy can be generated.

Option 4, writing to the disk and the lineprinter, is chosen whenever the user wants to generate a temporary copy for inspection.

Printing multiple copies option is chosen when the user is satisfied with the syllabus created, updated, edited and written by the Automatic Syllabus Generator.

CONCLUSION

The Automatic Syllabus Generator (ASG) can be very helpful to instructors preparing class syllabi. It can be used by a new instructor to create a syllabus containing consistent course material with what is being taught in other sections and in previous semesters. Instructors who have been teaching the course for several semesters who need to update their syllabi can also benefit. Although the ASG has some drawbacks, such as restricting academic freedom of the instructor, it should be noted that an instructor can still use the system and dictate most of the parameters. This can allow the instructor to take advantage of the time and economic savings produced by the ASG.

How Schools Use Microcomputers: Findings from the Johns Hopkins University National Survey of Computer-Using Teachers

Clarence Miller, Chair
Maryland State Department of Education
Baltimore, MD 21201

ABSTRACT

This symposium discusses the major findings and the implications for schools of a national survey of how schools are currently using microcomputers in their instructional programs. The survey, a national probability sample of 2,250 public, private, and parochial elementary and secondary schools, including about 1,400 with a microcomputer, was undertaken between December, 1982 and February, 1983. Respondents included the principal at each school and the primary computer using teacher in those schools with a microcomputer. The symposium consists of a presentation of some of the major statistical findings from the survey, and a discussion of the implications of these results for schools planning their future involvement with computers.

The paper presents a variety of data from the survey. The principal focus is how schools in different locations, having different types of student bodies, in different financial situations, and enjoying different degrees of district and administrative support have had correspondingly different histories of involvement with microcomputers, possess different types and quantities of microcomputer equipment, and use this equipment in different ways.

The paper also shows how schools that have had a microcomputer for over a year have changed their approach to using the computer and how these changes are related to factors in their environment.

The discussion which follows the paper presentation will present the reactions of two knowledgeable people in the field of educational computing - reflecting academic, governmental, and commercial perspectives. The discussants will focus on some implications of this national survey data for schools planning new or further investments in microcomputer technology in the services of their instructional goals.

PRESENTER

Henry Jay Becker, Project Director
The Johns Hopkins University
Baltimore, MD 21218

DISCUSSANTS

Arthur Melmed
U.S. Department of Educaton
Washington, DC 20208

Charles Blaschke
Educaton Turnkey Systems/MEAN
Falls Church, VA 22046

CAI in Foreign Language Instruction

Carl Adamson, Chair
Wichita State University
Wichita, KS 67208

Michael Bush
USAF Academy
USAF Academy, CO 80804

ABSTRACT

The mirocomputer offers foreign language educators a unique and powerful opportunity to enhance their language learning programs. Their interactive qualities provide students with feedback, evaluation and above all controlled stimulus-response learning which, when coupled with their impressive audio and graphic capabilities, may well make them an even more effective laboratory tool than audio tape. The first presentation offers a brief look at some of this potential with respect specifically to the Atari 800/400/1200 computers. After a brief discussion of some factors relevant for language laboratory implementation, the unique capabilities of the Atari will be presented. Features such as a modifiable character set, more than a dozen graphic/text modes, 256 color flexibility, graphics indirection through an easily modifiable display list, simultaneous audio, processing, and display, all lend themselves well to the highly interactive methodology of foreign language acquisition. In addition, the low cost and relative programming friendliness of the microcomputer make it particularly attractive for language learning applications.

The second presentation will discuss microcomputer based interactive videodisc in basic French instruction. The videodisc is the densest storage medium available for use on computers of any size. This new technology is being combined with the control capabilities of the microcomputer for the presentation of audiovisual material in basic language instruction. a

Using a methodology developed by an agency of the French Government during the past three decades, the Department of Foreign Languages at USAFA is studying how the "old" can be combined with the "new" to create an exciting environment for foreign language instruction. The random access capabilities of the intelligent videodisc system will be used to good advantage in the presentation of new material to beginning language students.

It is anticipated that by using this approach, students will individually accomplish the initial phases of learning new material, this freeing time presently being spent in the classroom on these more rote aspects of instruction. With the technology being used in the manner for which it is best suited, the classroom experience will be enriched for instructors and students alike and in ways possible only through the added human interaction that results from such use.

What Computer Curriculum Is Right For The Small College?

Dr. William Mitchell

The Univerity of Evansville
Evansville, Indiana

ABSTRACT

There are nearly 2000 colleges in the United States which enroll 5000 or fewer students. The majority of these colleges are liberal arts colleges with less than 2000 students. Most of these schools have acquired computer facilities and most offer one or more courses in which the computer is used. Many would like to make a computer-related major or minor available to their students. There are at least four nationaly published curricula which these school might consider but none of these curricula were formulated with the small college environment in mind. This paper will discuss the reasons why most small colleges cannot adopt any of the present curricula, and will offer a compromise curriculum which is a practical alternative.

Introduction

It is increasingly evident that computer applications will dominate our society during the close of the 20th century. Because of the pervasiveness of the computer phenomena, a field which is barely 30 years old is already fragmenting, ruptured by the stresses of compelling applications. As the academic community tries to understand what is cohesive and fundamental about computing and how to organize it for study, the various users of computing demand that greater emphasis be placed on those features of the subject which are immediately relevant to their individual needs. The economics of computing justify tremendous inefficiences in software and systems design because the hardware is so powerful that the resulting output is not only acceptable, it is significantly better than what can be achieved without computers. In a "results now" environment, there is little tolerance for abstractions and no motivation to take a little more time and do it right. In a rapidly changing technological environment, it is difficult to identify fundamentals. The computing field is surely unique, for it is an experimental field whose subject changes faster than it can be formalized. Just as we begin to understand sequential algorithms and automata, we find ourselves in the age of parallel processing.

The academic approach to computing reflects the diversity and fluidity of the field. The field is too young to have produced many great generalizers, so it lacks the unity provided by "schools of thought" which characterize other disciplines. There are no fundamental problems which challenge computer scholars (only an endless list of interesting or presssing ones), no widely accepted paradigms which guide their work [14,16], and consequently, little breadth to computer research. Like today's computer applications, the academic study of computing is particularized.

Little wonder, then, that so many definitions abound for undergraduate computing curricula. In the past five years the Association for Computing Machinery has published five different and discrete computing curricula, and these have been augmented by the recommendations of dozens of other professional societies and individuals, frequently without regard for any of the previous work. Aside from broadcasting the views of each constituency, the plethora of curriclum recommendations have begun, by their intersection, to describe what is desirable for undergraduate computing studies.

In this paper the author concentrates on four diverse curricula which have received national publicity in the professional literature. From these are drawn implications which guide the construction of a model curriculum for a small college. The method used to derive the model is as important as the actual curriculum derived. Each suggested curriculum represents a viewpoint, and from each's viewpoint, its curriculum is rational and the competitors' curricula are invalid. Thus, the derived small college curriculum will be viewed as sensible by those who accept the premises of the model's derivation, and will be viewed as inadequate by those who do not. If, however, the method for building the model is helpful, each may build his own ideal (for him) computing curriculum. This is not to say that computing curricula are arbitrary, but to emphasize the present lack of accepted generalizations.

The author can claim both knowledge and experience in the process of curriclum design [8,9], and a special expertise in the area of small college computing curricula [17]. Many of the ideas presented here have emerged from a graduate course on computing curricula taught to small colllege computer educators. The curriculum derivation process has been employed by dozens of colleges in the past five years.

The Curricula

The major undergraduate curriculum models for computing education have been proposed by professional groups, each with its own bias. Three curricula are proposed by the Association for Computing Machinery: Curriculum '78 in Computer Science [2] (see figure 1), the 1982 Information Systems Curriculum [12] (see figure 2), and the Two-Year College Curriculum of 1980 [8]. The first curriculum represents computing as it exisc in the research-oriented university. Emphasis is placed on the mathematical analysis of algorithms, on the theory of numerical computation and symbol processing, and on the study of the properties of abstract machines and languages. The second curriculum emphasizes the application of existing computer technology to business organizations. Concern is evidenced for the effective use of computer systems, which necessitates understanding how information is used for control and decision making within organizations. The third curricula is focused on the operation of computer systems in commercial environments with the goal of preparing students for carrying out the activities normally required in such shops (rather than to analyze why these activities are desirable). Both of the later two curricula emphasize a strong knowledge of business as corequisite to utilizing the technical content of the computer curriculum. All these curricula have been devised by academics with the advice and counsel, to varing extents, of the practicing professionals.

Two other curricula have been derived to represent the business data processing community. The DPMA curriculum [1] (see figure 3) eschews theory, even the business organizational theory emphasized by ACM, and focuses instead on criteria for practice. The curriculum seeks to produce a competent employee who will be skilled in applying existing computing technology to the present-day business environment. The highest goal of the curriculum is to minimize the time it requires to assimilate the graduate into his first position. To meet this goal the DPMA curriculum designers were willing to sacrifice "why" for "how" when that was necessary. The curriculum is implemented as a rolling five-year plan, so that each graduating class will be state-of-the art. It is thus deemed essential that computer technology be highly integrated with courses in business practice in order that the student be properly oriented and motivated.

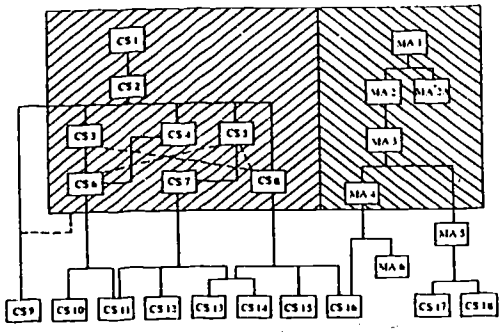
The curriculum of the Pittsburgh Large Users Group [13] (see figure 4) has been constructed similarly to the DPMA model, utilizing a survey of professionals to ascertain topics and the degree of emphasis. In this case, the survey was limited to large-scale corporate computer centers, so that the result supported greater technical depth than the DPMA survey, reflecting the greater technical sophistication of such centers. While encouraging the business corequisites of the other commercially-oriented curricula, the PLUG curriculum does not insist on the degree of integration espoused by DPMA. On the other hand, while it recommends technical courses similar in depth to the ACM Information Systems proposals, it does not seek the improvement of management's utilization of computer systems. Its goals is to provide the technically competent employee which the two year curriculum attempted to define, except that in this environment the four year degree is necesary to achieve that competence.

Another nationally publicized undergraduate computing curriculum was proposed by the IEEE Computer Society in 1977 [4], intended to guide the offering of computer engineering within engineering schools. The similarity between this curricula and Curriculum '78 has been analyzed [3] and found to be substantial, but this curriculum offers little else to non-engineering programs.

Each of the four-year curriculum models are structured with a core of required courses and a range of elective courses at various levels. The Computer Science core is eight courses intended to be pursued mostly in the lower division. ACM's information system's curricula has an eight course core which is taken mostly in the upper division while the major is completing the AACSB common body of knowledge (out of which a statistics course and a programming course are prerequisite to the computing core). The DPMA curricula has a seven course core, four lower division and three upper division, while the Pittsburgh curriculum has an eight course core, two lower division and six upper divison. The DPMA curriculum is unique in being designed with the intention that its lower division courses would frequently be taught in junior colleges. The author has elsewhere analyzed the DPMA curriculum in the context of its stated toals [10] and the similarities of the DPMA and ACM Information Systems models and the dissimilarities of the DPMA and Curriculum '78 models have been noted [15,6].

The Computer Science curriculum lists ten upper division electives and assumes that a major will take at least four. It is also expected that at least five mathematics courses will be taken (to include calculus,

Figure 1. ACM Curriculum '78



COMPUTER SCIENCE MODEL CURRICULUM

Core Courses:

CS 1. Computer Programming I
CS 2. Computer Programming II
CS 3. Introduction to Computer Systems
CS 4. Introduction to Computer Organization
CS 5. Introduction to File Processing
CS 6. Operating Systems and Computer Architecture I
CS 7. Data Structures and Algorithm Analysis
CS 8. Organization of Programming Languages

Elective Courses:

CS 9. Computers and Society
CS 10. Operating Systems and Computer Architecture II
CS 11. Data Base Management Systems Design
CS 12. Artificial Intelligence
CS 13. Algorithms
CS 14. Software Design and Development
CS 15. Theory of Programming Languages
CS 16. Automata, Computability and Formal Languages
CS 17. Numerical Mathematics: Analysis
CS 18. Numerical Mathematics: Linear Algebra

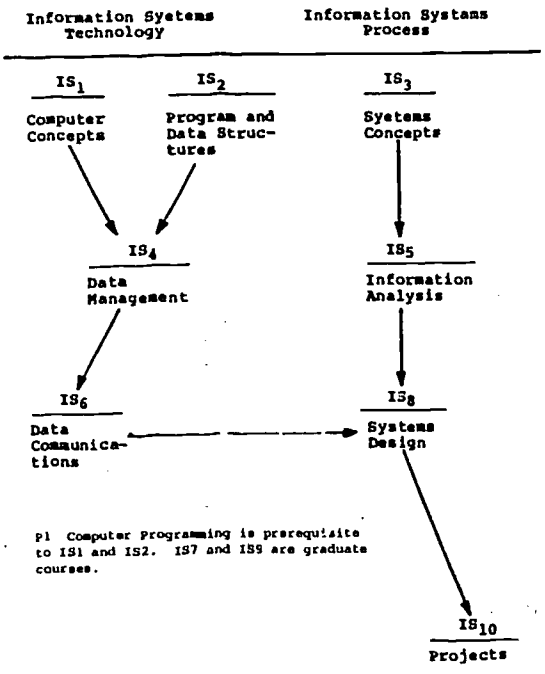
Special Topics Courses:

A. Microcomputer Laboratory
B. Minicomputer Laboratory
C. Performance Evaluation
D. Telecommunications/Networks/Distributed Systems
E. Systems Simulation
F. Advanced Systems Programming
G. Graphics
H. Compiler Writing Laboratory
I. Structured Programming
J. Topics in Automata Theory
K. Topics in Compatibility
L. Topics in Formal Language Theory
M. Simulation and Modeling

Mathematics Requirements:

MA 1. Introductory Calculus
MA 2. Mathematical Analysis I
MA 2A. Probability
MA 3. Linear Algebra
MA 4. Discrete Structures
MA 5. Mathematical Analysis II
MA 6. Probability and Statistics

Figure 2. ACM Information Systems Curriculum



P1 Computer Programming is prerequisite to IS1 and IS2. IS7 and IS9 are graduate courses.

Figure 4. PLUG Curriculum

BUSINESS INFORMATION SYSTEMS CURRICULUM STRUCTURE

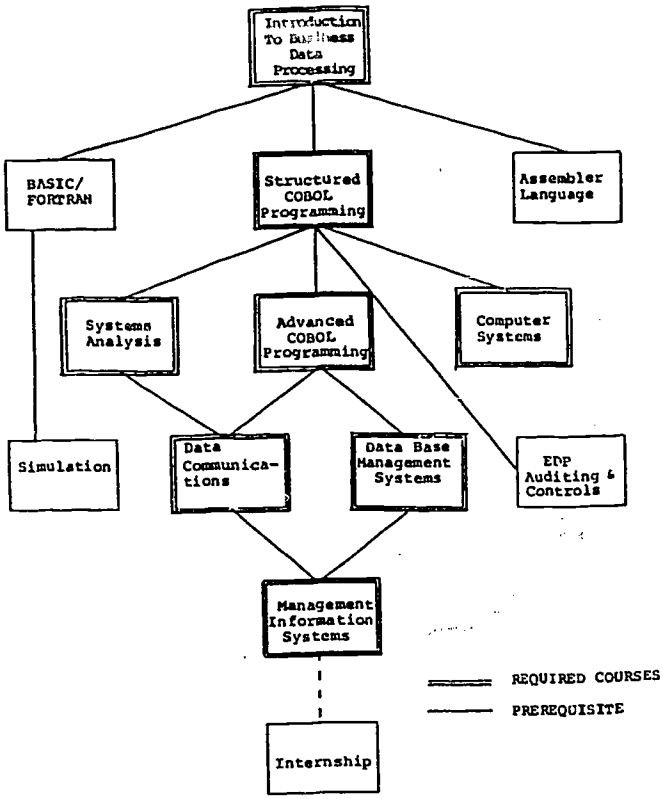
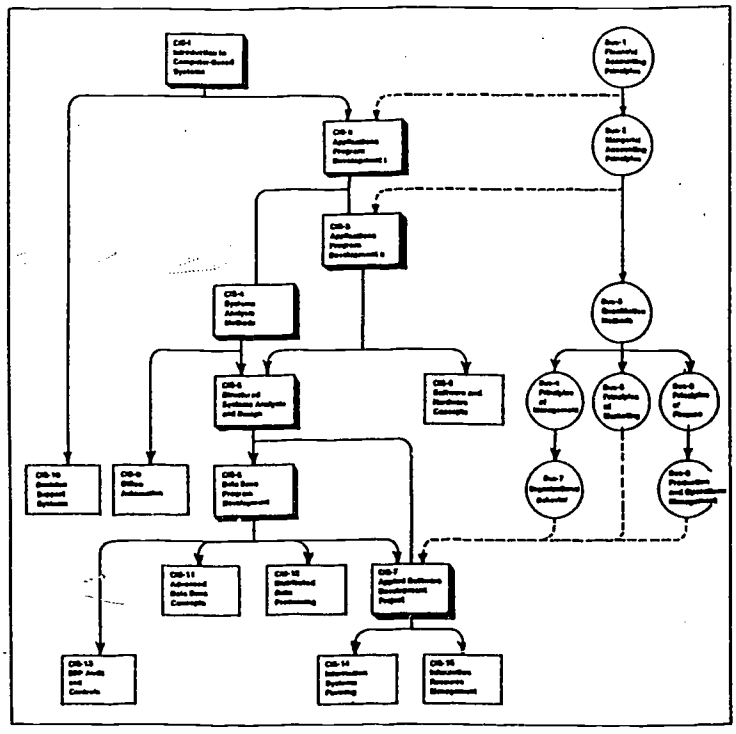


Figure 3. DPMA Computer Information Systems Curriculum



probability, linear algebra and descrete structures), and several courses in an applications area, such as the physical or business. The Information Systems lists no electives. The DPMA calls for three upper division electives to be chosen from a list of eight, and suggests eight corequisite business courses in lieu of the AACSB core (including statistics). The Pittsburgh curriculum suggests two electives from a list of five, two of which are lower division, and it requires eight business courses, two business electives and five mathematics courses (including statistics and business calculus).

All of the curricula are characterized by deep prerequisite chains (up to six levels) and by the recommedation of an upper division software project course. A major consists of nine 3 semester hour computing courses for the DPMA and ACM Information Systems curricula (discounting the general education course CIS 1 and including the prerequisite programming course P1), ten for the PLUG curriculum, and twelve for Curriculum '78. At least half of the major consists of upper division courses, and none of these are language-oriented programming classes (learning to program in a high level programming language is universally considered a lower division task, but several languages are expected to be mastered).

The Small College Environment

The liberal arts college finds all four curricula unimplementable for a variety of reasons. The Computer Science curricula is too sophisticated in its present emphasis, and there are pressures to make it even more abstract [11]. Curriculum '78 strives to transcend the practical, but to do so requires a theoretical knowledge which is not available to the small college. Even though most of the computing programs in the small college will emerge from the science or mathematics departments, the formality and abstractness of computer science is yet several steps removed from where the small college mathematics or science faculty are. Computers are relatively new in these schools and great effort is being expended just to gain concrete experience with their use. This experience base is too meager to bear the weight of a theoretical curriculum. The average small college faculty would be pressed to do a competent job presenting all of the eight core courses of Curriculum '78.

On the other hand, the three undergraduate business curricula vary from too narrow and vocational (presupposing an environment which cannot be approximated at the small college) to too narrow and sophisticated (presupposing a computationaly trained business faculty). Specialization, either in the direction of computer hardware and software (telecommunications, data base, large scale systems) or in the direction of operations research, the automated office, DP law or computer auditing is beyond the resource of the small college. The average small college has mini- and micro-computer systems, little or no data processing expertise, and its computing knowledge and business knowledge reside in separate individuals.

In these circumstances the small college must find a curriculum compromise between the mathematical and the commercial, between the abstract and the applied, which will be within the skills of the faculty to deliver, yet will effectively prepare its students for computing careers. What these students will lack in technical sophistication and state-of-the art integration, will be balanced, the liberal arts college believes, by a broader perspective on humanity, society, culture and learning. Rather than having specialization in the environment they intend to enter, the liberal arts student will be prepared for a more general context of life. The thesis is that while they will still have much to learn about either data processing or computer science, they will be prepared to learn quickly, and they will organize their knowledge within much broader perspectives, enabling them to exercise more humane judgement in their work.

Rather then attempt to persuade the small college that this thesis is incorrect and that they should seek to emulate either the business school or the research university, we present a rationale for a curriculm which meets thier perceived needs.

A Small College Curriculum

The small college is advised to offer courses at various levels and in various tracks so as to satisfy the needs of its hetrogeneous student body. The small college must be equally concerned with the availability of opportunities for computer literacy, for two and three course "minors" for students from a variety of disciplines (principally from business and mathematics), and for a major. The curriculum models all take the narrow view of the specialized major, so they give no guidance on how to best serve this complex goal. But since each does speak to the introduction of the major to the field, we can compare their approaches.

The DPMA model has the weakest introductory course in terms of requirements, or, if you prefer, it suggests the broadest introductory course. The commercially-oriented curriculums in general presuppose the existing practice of using as the first course in the major the "Introduction to Data Processing" course which was developed as a core course for all business students. The DPMA model further suggests that this course should cater to non-business students who are seeking an introduction to computing for general

education. In the DPMA model, programming training begins in CIS 2.

The ACM Information Systems curriclum and the PLUG curriculum assume a more rigorous, programming-oriented first course, but still share with other business majors. The PLUG curriculum specifies BASIC the first course language, but in the ACM Information Systems model, P1 is modeled on the first course of Curriculum '78, and requires a procedure-oriented language which will support later programming experience. Curriclum '78 requires an in depth first course in a high level programming language which stresses methodology and algorithm development.

It is obvious that a single course in the small college cannot meet all these thrusts. Therefore, at the freshman level we propose an introductory programming course, an introductory systems analysis course and a computer literacy course, each taught without prerequisite. Instead of combining an introduction to programming with a narrative about DP, we suggest two separate courses which will allow greater depth in each. To meet the needs of the poorly prepared or weakly motivated student, the literacy course would provide opportunity to be exposed to programming and deriving algorithms, but would not make that aspect of computer interaction a major feature of the course. The literacy course might use either a Computers and Society text or an introduction to Data Processing text, but its focus is fundamentally non-major. Ideally it should incorporate extensive interaction with computers as media for entertainment, for CAI, for word processing, for library use, for data base query, and for packaged functions like statistical analysis or spreadsheet modeling.

The sophomore courses, roughly following the three tracks of the PLUG curriculum and the ACM Information Systems model, should be a computer systems course, a second programming course, and some elective language courses (taught as second languages). The goal here is to provide both the necessary development of the major and second courses for various types of minors. The business minor, after one systems course (with or without the literacy course), would take the first programming course and then a business language course (COBOL). The mathematics major, on the other hand, might move from the first programming course to the second, or to FORTRAN or some applied mathematics course without ever taking the systems course.

The junior level courses should include a data and file structures course, the second systems course, and a topics course which would be a proving ground for new courses. The second programming course would be the prerequisite for both these courses, and one or both of the systems courses might be prerequisite to the topics course. One of the second languages courses, which, like the topics course, should alternate content each year, might be taken simultaneously with the topics course. It is possible, therefore, for a student to break the lockstep of the schedule if necessary.

The senior level courses will include a data base course, a simulation course and an application project course, and, if possible, a second upper division elective. Both the simulation course and the data base course make extensive use of the data structures course, and more than casual use of the computer systems material. The resulting curriculum bears greatest similarity to the ACM Information Systems requirements, but in spirit it is closest to the PLUG curriculum. Elective courses might be developed to provide the Curriculum '78 software and hardware topics which have been squeezed into the computer systems course (course content is discussed below) but the presumption is that most small colleges will never be able to offer the number of courses required by Curriculum '78 and still have the hardware and faculty resources to meet the computing education needs of the nonmajors. It is also presumed that students will not choose the small college with the intention of becoming a designer of operating systems or the builder of pipeline processors. More likely, the student will be content to pursue the details of computer science in graduate school, or will be intending to apply his computer knowledge vocationally immediately upon graduation.

The core of the small college major would consist of Programming I and II, Systems Analysis I and II, Computer Systems, Data and File Structures, and the senior level project course. A major would require, in addition, a lower division elective and two upper divisions electives. In the following paragraphs we discuss each course and its role in the curriculum. The reader is directed to the appendix to see the course descriptions of the courses referenced from the four models.

Course Descriptions

Programming I and II should be taught in a block-structured procedure-oriented language and should be a "no nonsense" programming course akin to the Curriculum '78 courses CS1 and CS2. The first course will introduce decision structures, looping structures (nesting), data types, lists, tables, and subroutines. Standard file processing activities should be illustrated (such as contol break processing and sequential update). Proper techniques for documenting software and for developing programs should be taught and consideration given to matters of style and group programming.

The second programming course should systematize and extend the experience of the first course in the same language. The student is introduced to more sophisticated

use of data structures and information coding techniques. The emphasis is on the variability of data, complex data organization, and interrelationships between data structures. Preview linked lists, trees, graphs and recursion. Expose the student to the analysis of algorithm efficiency, demonstration of correctness (assertions and testing methodology), and searching and sorting principles. Do a large group project which requires module design and interfacing, and [illegible] decomposition of problems and typical [illegible] structures.

The programming sequence is intentionally experience-oriented, building experience within a language and experience in problem solving. The programming exercises should dominate the students course requirements and should be rigorous enough to discourage those who resist practicing the tools being presented to manage complexity. High standards of internal and external documentation and user friendliness should be required of all software products.

Since this sequence must meet the needs of both scientific and commercial programmer/analysts, it should focus on good coding practices and illustrate the broadest range of techniques. It should not be taught in FORTRAN, BASIC or COBOL if possible, though BASIC could be utilized if extreme care were taken to simulate structured control sequences. (BASIC, like PL/1, has gained adherents in both the scientific and commercial fields, hence is justifable to some extent in terms of the generality of its potential utility). Students emerging from the sequence will understand the practical process of software development and will know one language very well. Only the first course in the sequence is required before electing to study computer systems or a second language.

The systems analysis series begins with a survey course which touches lightly on computer hardware technology, but treats in depth the software life cycle, the organization and interaction of data processing with users, and the role and tools of the systems analyst. This course could serve as an elective for business-oriented students. Because of the great amount of overlap between CIS 4 and CIS 1 in the DPMA curriculum, this first systems course could cover the topics of CIS 4 from their introduction in CIS 1, but not reach as high a level of skill development as is published for CIS 4. The second systems course would be modeled on CIS 5.

The Computer Systems course could be modeled on the Pittsburgh course with that title, or on the Information Systems curriculum's IS1, or on DPMA's CIS 8, but should include a software project in the same language used in Programming I. This course, along with the programming sequence is prequisite to Data and File Structures. The Information Systems course IS2 provides an adequate topic outline for the Data and File Structures course. The course should introduce a file oriented language if a language such as ADA or PL/I is not already in use. The programming exercises should illustrate sophisticated applications of file structures and lay a firm foundation for the data base course.

The data base course would be a notch above IS4 and several notches above CIS 6 or the Pittsburgh model's BIS 401. Previous exposure to the data structures and file structures of DBMS packages will permit greater attention to both the theory of data base architectures and to the context in which they are used. It is unlikely the small college will have access to IMS or TOTAL, but inexpensive network and relational-like data base packages exist for micros and most minis, such as HP and DEC. Thus a blend of theoretical exposure and practical experience could reasonably be achieved in this senior level elective. In some environments CS 11 would be an appropriate guide, but it is oriented toward the writing of a DBMS rather than toward its use in solving applications.

The simulation elective could be modeled on BIS 410 of the Pittsburgh curriculum or on IS7 of the Information Systems curriculum. As in the case of data base, simulation languages such as NDTRAN or SLAM are readily available on mini-computers. Without prejudice to either major, the data base course might be more attractive to the commercial programmer/analyst, and the more mathmatically oriented simulation course could interest the scientific programmer. Hopefully both groups would see the relevance of both electives.

The lower division elective courses would normally be second languages, and would provide intensive syntax-focused introductions to the problem demains appropriate to the language. Thus the COBOL syntax covered in DPMA's CIS 2 and CIS 3 would easily be covered in a single sophomore level course after experiencing Programming I. Likewise a FORTRAN course oriented about numerical methods could be offered, or a comprehensive BASIC course, or RPGII or assembler, etc. The upper division topics course would deal as appropriate with comparative computer languages, with a second "advanced programming" course in COBOL, with data communications, with computer architecture, or with Management Information Systems.

The capstone course for the major is the senior level application project course which should require that the spectrum of analysis, design, coding, and documentation activities be applied to a realistic problem and usually be carried out by a small team. This course will test the graduate's preparation to perform in the area of software development. CIS 7 or IS10 are appropriate models.

Some colleges are experimenting with an upper division social issues course which is open to either majors or nonmajors. Such a course could have as prerequisite as little as the freshman literacy course, or as much as both of the first courses in programming and systems. CS 9 could be helpful, though it envisions a still more sophisticated audience.

Assuming that the normal faculty load is four courses a semester, this curriculum could be offered by one full-time computer faculty member (teaching the core) supported by a part-time business faculty and a part-time mathematics or science faculty member (or several adjuncts) for a total of 2 FTEs (see figure 5). Such a load is not desirable, and sections would have to be kept small in order to keep student contact hours tolerable (computing is much more student intensive than is mathematics or history [7]). Yet the national faculty situation is such that expecting every small college to have access to two well-credentialed computing instructors is also unrealistic.

A very respectable systems information major is produced if the student completes the core, elects the data base course, the lower division COBOL course (assuming the programming course is in Pascal), the upper division topics course in advanced COBOL applications (with the COBOL course being taught by the business faculty member or adjunct), and completes a minor in the business department. Such a student would compare very favorably (having greater technical skills) with a DPMA graduate who had elected CIS 8, CIS 11, and CIS 13 or CIS 14. Such a student would have programming skills exceeding that acquired in the Informations Systems or Pittsburgh curriculum, but would lack exposure to data communications and to management information systems.

Figure 5. Small College Curriculum Schedule

Fall Semester	Spring Semester
freshman courses	
Literacy Systems I	Literacy Programming I*
sophomore courses	
Programming II*	Language elective Computer Systems*
junior courses	
Data and File Structures* Systems II*	Topics*
senior courses	
Data Base elective	Simulation Project*

*signifies a course taught by full-time computer faculty (others may be adjuncts)

A students with mathematics or science interest could also pursue the core courses, and could achieve, with an alternate set of electives, a competent exposure to the core concepts of Curriculum '78, even though the offerings would be too meager to have ventured far outside that core.

Conclusion

Despite the obvious biases present in each of the curriculum recommentations, it is possible to recognize a collection of topics which are common to those models which emphasize computer applications. Not surprisingly, much of this intersection is also contained in the computer science core. Thus the courses described for a small college can capture a high percentage of the topics advocated by any of the models, and do so while emphasizing the fundamentals of multi-lingual programming. The graduate of the small college, regardless of the computer equipment he trained on, can reasonably expect to be competitive with graduates of any of the other curriculum models for the entry level programming position. The liberal arts graduate will bear the responsibility for doing a great deal of integrating on his own, and the obligation to learn on the job new hardware and software details. But since these hardware and software details are constanty changing anyway, and since even large universities are unable to maintain a computing environment commensurate with industry, these handicaps may well prove to be virtues. The liberal arts student is prepared to change, and forced to imagine different environments, where the DPMA student may well be too specifically oriented. By concentrating on technical fundamentals rather than specific business applications, the small college faculty can educate both the commercial and the scientifically oriented student in the same core courses, and limit to the elective courses the need for multi-disciplinary expertise.

References

1. Adams, David R., and Thomas H. Athey, DPMA MODEL CURRICULUM FOR UNDERGRADUATE COMPUTER INFORMATION SYTEMS EDUCATION, DPMA Education Foundation, Park Ridge, Illinois, 1981.
2. Austing, Richard, et. al., "Curriculum '78, Recommendations for the Undergraduate Program in Computer Science," COMMUNICATIONS OF THE ACM, v. 22, n. 3 (March 1979).
3. Engel, Gerald L., "A Comparison of the ACM/C3S and the IEEE/CSE Model Curriculum Subcommittee Recommendations," COMPUTER, v. 10, n. 12 (December 1977).
4. IEEE Computer Society Education Committee, Model Curricula Subcommittee, A CURRICULUM IN COMPUTER SCIENCE AND ENGINEERING, 1977.
5. Jones, Ron, and Rich Hamilton, "Computing Education, the DPMA Model,"

COMPUTERWORLD, v. XV, n. 38 (September 21, 1981).
6. Kroenke, David, "A Place in the Sun," INTERFACE, v. 3, n. 1 (Spring 1981).
7. Little, Joyce Currie, RECOMMENDATIONS AND GUIDELINES FOR AN ASSOCIATE LEVEL DEGREE PROGRAM IN COMPUTER PROGRAMMING, ACM 1981.
8. Mitchell, William M., THE DESIGN OF MATHEMATICS CURRICULA FOR THE SMALL COLLEGE, Ph.D. Dissertation, George Peabody College, 1974.
9. Mitchell, William, and Bruce Mabis, "Implementing a Computer Science Curriculum Merging Two Curriculum Models," SIGCSE BULLETIN, v. 10, n. 3 (August 1978).
10. Mitchell, William, and James Westfall, "Critique and Evaluation of the CAL POLY/DPMA Model Curriculum for Computer Information Systems," SIGCSE BULLETIN, v. 13, n. 1 (February 1981).
11. Mulder, M. C., et. al., "Computer Science Program Requirements," a position paper of the Joint ACM/IEEE Task Force for Computer Science Program Accreditation, February 14, 1983.
12. Nunamaker, Jay F., Jr., et. al., "Information Systems Curriculum Recommendations for the 80s: Undergradute and Graduate Programs," COMMUNICATIONS OF THE ACM, v. 25, n. 11 (November 1982) (summarized in COMPUTERWORLD, v. XV, n. 39 (September 28, 1981).
13. Schultz, Brad, "Model DP Curriculum Welcomed by Colleges," COMPUTERWORLD, v. XV, n. 40 (October 5, 1981).
14. Traub, J. F., Editor, "Quo Vadimus: Computer Science in a Decade," COMMUNICATIONS OF THE ACM, v. 24, n. 6 (June 1981).
15. Vanecek, Michael T., and Carl Stephen Guynes, "Business Computer Information Systems DPMA, vs ACM: Now What?" INTERFACE, v. 3, n. 4 (Winter 1981-82).
16 Zant, Robert, Michael Vanecek and Carl Guynes, "Thoughts on the Maturing Process of the Information Systems Academic Discipline," INTERFACE, v. 4, n. 2 (Summer 1982).
17. Zientara, Marguerite, "University Summer School Retraining College Professors to Teach Computer Science," COMPUTERWORLD, v. XVI, n. 18 (May 3, 1982).

A New Source of Computer Science Teachers:

Faculty Members From Other Departments

Keith Harrow

Department of Computer and Information Science,

Brooklyn College, Brooklyn, N.Y. 11210

Abstract

The current shortage of computer science faculty is well-known. This paper describes a novel solution to the problem that has been developed at Brooklyn College. Faculty members from other disciplines are retrained to teach introductory computer science courses. In addition to fulfilling our basic need to staff courses, this approach has a number of interesting ramifications, both good and bad.

Introduction

In the past few years, almost all colleges and universities have experienced a shortage of qualified computer science faculty. Many people have explored the implications of this problem and offered a few long-term solutions [1, 2, 3, 5]. At the Thirteenth Annual SIGCSE Symposium on Computer Science Education, there were a number of papers and panel sessions on ways to attract and retain computer science faculty.

The Brooklyn College Problem

At Brooklyn College, we are experiencing the same difficulties as everyone else, with a few extra local problems. Over the past 10 to 12 years, the college enrollment first almost doubled to 30,000 full-time students, then contracted to its present level of about 15,000. The drastic changes in the size of the student body, plus significant shifts in enrollment from one area to another, have left Brooklyn College with a number of overstaffed departments. For example, the Chemistry Department has over 30 tenured faculty members, but relatively few courses for them to teach. Recently, there has been an increase in the need for teachers of remedial courses, especially Mathematics and English. The Computer and Information Science (CIS) Department has experienced an explosive growth, going from three or four computer sections in 1970 to the current total of more than 100 sections. The computer science program is particularly rich, including introductory courses (in PL/I for those interested in an intensive programming course, and in Basic for those interested in just a brief introduction), a large number of advanced electives and a growing graduate program.

The CIS Department has 19 full-time faculty members, which is about half the number that would be justified by our enrollment as a percentage of the total college enrollment. Even worse, many computer science faculty members do not have a full teaching load; most people are released from one or more courses because of administrative duties and/or research work. We have been relatively successful in attracting new faculty, but most of them have been research-oriented, with very low teaching loads. These people contribute enormously to the department in terms of grants, publications, and prestige; but they do not contribute in any significant way to our teaching power.

Thus, computer science faculty members teach only about forty percent of all computer science sections, with the majority of these being graduate and advanced undergraduate courses. In the past, we have used part-time or adjunct faculty members to fill the gap. However, the shortage of full-time graduate students, plus the college's lack of money, have combined to limit our ability to use adjunct lecturers.

To summarize, the problem that we (and any number of other schools) face is the following: given these constraints, how do we meet the increasing demand for computer science courses?

Possible Solutions

There are a number of unattractive solutions to this problem. One is to limit enrollment in computer science courses (many schools have adopted this approach). For obvious reasons, the Brooklyn College Administration does not like the idea of turning away hundreds of potential students. Another idea is to use large lecture sections in the introductory PL/I course. We have tried this method and found it inferior to our current format (approximately 25 sections with 25-45 students per section).

A third idea has gradually evolved over the past few years. This solution solves the immediate problems of relieving the overstaffed departments and covering computer science courses. It also raises a number of provocative questions for the future.

The solution that we have developed uses faculty members from other departments to teach computer science courses. To some degree, this crossover from one department to another is not new. For example, a chemist might teach a course in an area of specialization (e.g., real-time systems); a mathematician might teach a course in numerical analysis; a linguist might teach a course in formal language theory. Most computer science departments probably have such informal arrangements with other departments. However, our use of outside staff is much more extensive.

In the Fall 1982 semester, 13 faculty members from other departments were teaching one or more computer science courses. More than half the sections of the introductory programming course in PL/I (plus a few intermediate electives) were taught by these people, with adjunct faculty teaching most of the remaining sections (mostly at night and on the weekends). Currently, we have faculty members from the Departments of Chemistry, Educational Services, English, Mathematics, Physics, and Psychology teaching computer science courses. Approximately one fifth of all computer science courses are taught by full-time faculty members from other departments. It is important to note that these people maintain their positions within their own departments, but teach computer science courses to fill out their teaching loads.

Faculty Development Program

Of course, we are unwilling to accept just any faculty member from another department. The Brooklyn College Administration has agreed to give the CIS Department Appointments Committee the right to interview and approve all such candidates. Naturally, we are reluctant to accept a faculty member's self-assessment of his or her competence to teach computer science. Therefore, we have made a major effort to certify some of these people.

For the past three summers, the CIS Department (under the auspices of the Vice President for Academic Affairs) has organized a Faculty Development Program. In 1980, Professor Pat Sterbenz conducted a small test project to teach PL/I to those with some previous knowledge of programming (typically Fortran). In 1981, the Chairman of the CIS Department, Professor Frank Beckman, proposed a more ambitious program, one part of which was designed to teach other faculty members how to teach computer science. With the aid of a number of my computer science colleagues, I supervised the Faculty Development Program in 1981 and then again in 1982.

A detailed description of the 1981 program is given in [4]. Here is a brief overview (the 1982 program was similar). All participants were expected to have some previous knowledge of PL/I (in practice, this was not always true). The faculty members were asked to audit the equivalent of a second course in PL/I, even though they would be teaching an introductory course. Thus, they would be exposed to more of computer science than they were expected to teach. In addition, weekly meetings were held to discuss the style and presentation of material, problems students seemed to be having, and differences between teaching computer science and teaching their own disciplines. They were required to complete all programming assignments for the course (but they did not take any exams).

Current Status of the Program

After successfully completing the Faculty Development Program (some did not complete it), a faculty member is accepted as a candidate to teach CIS 1, our introductory PL/I course. The department then provides the instructor with a syllabus, several different course outlines (corresponding to different approaches to teaching the course), model homework assignments and exams, etc. Each new instructor is observed informally two or three times per semester, until we are confident that the instructor is doing a good job and needs less supervision. The CIS Department Appointments Committee reviews all new instructors every semester, and has the right to reject any candidate. We discuss such matters as their progress in learning more computer science, observation reports, and other feedback on their teaching.

Faculty members are urged (but not quite ordered) to continue their education in computer science by taking further computer courses. It is suggested that they eventually learn the equivalent of at least three or four courses, including assembly language programming, data structures, and programming languages. Although a few people have been somewhat remiss, most of the new faculty members have been extremely conscientious. As part of the faculty union's collective bargaining agreement, faculty members receive a full tuition waiver when they register for graduate or undergraduate courses. Many people have taken advantage of this offer and received credit for courses in such areas as Cobol programming, file processing, compiler construction, simulation, and so on. (One graduate of the 1981 Faculty Development Program has made so much progress that she has shifted her department affiliation - because of budget cutbacks, she was about to let go by her old department. She is currently an instructor teaching a full load in CIS.) We have then asked some of the more advanced people to teach second or third level CIS courses, and they have done quite well. Thus, as we produce new candidates to teach introductory courses, we hope to move the more experienced teachers into intermediate-level courses.

Evaluation of the Program

As with most experiments, there are both good and bad things to report about our use of faculty from other departments. Let's start with the negative points.

The most obvious problem is that these faculty members are not computer scientists. Even if they do a good job in teaching little details, they do not always see the global point of view. In addition, they tend to be less sensitive to some of the things that we consider to be important (e.g., style of programming).

This is especially true for old Fortran programmers who have a lot of bad habits. We are trying to solve these problems by requesting these people to broaden their exposure to computer science, by asking them to study other aspects of the field (e.g., assembly language or data structures), and by emphasizing to them what we consider to be important. In particular, by having them take a second or third course after the introductory one, we hope that they will understand more clearly what a student who has completed the first course should know.

Another potential problem that we were concerned with was the development, especially in the Administration, of a false impression that anyone can teach computer science. Fortunately, in practice this has not been a problem. We have continually emphasized the need for special training and for a final review of all candidates by our department. So far, we have been successful in resisting the temptation to flood CIS with a horde of new instructiors, and we have been able to assert our authority by rejecting certain candidates because of a lack of preparation.

However, there are real problems with supervising people once they are teaching. Our department has a preponderance of young faculty members, especially in comparison to the more established departments. It is quite hard for an untenured faculty member (or a graduate student or an adjunct lecturer) to criticize a senior faculty member from another department. Thus, our chairman and one or two other senior computer scientists have been forced to serve as intermediaries in a number of delicate situations.

The problem that we are most concerned with is related to this. How can we eliminate someone from another department who is doing a poor job? We have one person who seems to be relatively weak as a computer science teacher (he may very well be weak as a teacher in his own department). We have made a number of suggestions to him, but with little effect. As of now, we would rate the job he is doing as adequate, so the problem is not really acute. However, we may one day be faced with a situation in which an instructor is doing an unacceptable job. Will we be able to remove such an instructor?

Despite the criticisms mentioned above, we are pleased with the results of the program. First, our introductory courses are being covered in a reasonable way, enabling us to maintain an assortment of advanced undergraduate and graduate courses. The instructors from other departments are all experienced teachers, unlike most adjunct lecturers who are usually full-time programmers, not teachers. Thus, we do not have to worry about missed classes, unprofessional conduct, etc. These faculty members are all familiar with Brooklyn College students and academic regulations. We encourage them to attend our department meetings and they have provided a number of interesting perspectives on some important issues.

Second, we have gained a number of new friends in other departments. Most of the retrained faculty members are quite happy to be teaching computer science to motivated, intelligentstudents (for many, the alternative is teaching remedial mathematics). It is also good for them professionally, since a knowledge of computer science will almost surely be helpful as a tool in their own disciplines. (It will be interesting to see if any novel uses of a computer in other fields of research are introduced as a result of our program.) Teaching in our department has made them more aware of many of the CIS Department's special needs - e.g., the continual need to upgrade equipment. In many cases, they have served as our spokesmen on college-wide committees investigating these needs. Most of them have gained an increased appreciation of computer science as a legitimate academic discipline. As noted above, these faculty members have maintained their positions within their own departments. We hope that they will share their increased awareness of computer science with their colleagues.

Finally, we have shown the Brooklyn College Administration that we are trying to help the college as a whole. In a time of severe budget cuts and talk of dismissal of all untenured staff, we have made a good-faith effort to help solve these problems. The college recognizes these efforts, and has been generous in meeting many of our other requests, including the hiring of several new computer science faculty members. It would be easy to insist that the CIS Department needs five new full-time faculty members per year. But given the current demand for computer science faculty, it would be quite hard for us to find five competent people each year. By retraining faculty members from other departments, we have been able to concentrate on finding one or two good people per year and we are able to maintain the quality of our faculty.

Summary and Prospects for the Future

In summary, we are pleased with the job being done by most of the faculty members from other departments. To repeat, we do not view them as permanent substitutes for legitimate computer scientists; but we do accept them as short-term replacements. Many colleges and universities either already face or will soon be facing similar staffing problems. We hope that our program can be used as a model at other institutions.

References

1. Peter Denning, "ACM President's Letter; Eating Our Seed Corn" Comm. of the ACM 24, 6 (June 1981), 341-343.

2. Peter Denning et al, "A Discipline in Crisis: The Snowbird Report", Comm. of the ACM 24, 6 (June 1981), 370-374.

3. Gerald Engel and Bruce Barnes, "Employment Decisions by Computer Science Faculty: A Summary of the 1980-81 NSF Survey", Proc. of the Thirteenth SIGCSE Technical Symposium on Computer Science Education (Feb. 1982), 167-169.

4. Keith Harrow, "A Faculty Development Program", Proc. of the Thirteenth SIGCSE Technical Symposium on Computer Science Education (Feb. 1982), 170-173.

5. J. F. Traub, "Quo Vadimus: Computer Science in a Decade", Comm. of the ACM 24, 6 (June 1981), 351-369.

HOBBY ROBOTS AS TEACHING/LEARNING TOOLS

Michael Moshell
The University of Tennessee
Knoxville, TN 37996-1301

Charles Hughes
The University of Central Florida
Orlando, FL 32816

Carl Gregory, Lee Wittenberg
Gentleware Corporation
Knoxville, TN 37919

Abstract

The development of hardware and software for simple robots is proposed as a follow-on experience for students who have completed an introductory computer programming course or tutorial.

Goals and a syllabus for a course, equipment and software needs and resources are described. Emphasis is on easily available, inexpensive resources and the use of microcomputers as control devices.

The course is intended to teach principles of physical mechanics, cybernetics, computer science, and to develop problem solving skills.

The course of study being designed is appropriate for both individual and class use. The course is embodied in a tutorial book, and is supported by a software package and a hardware computer interface for robotics experiments.

Outline

1. Goals of the Course
2. Syllabus for the Course
3. Hardware for Hobby Robotics
4. Software Concepts and Tools
5. Speculations about Impact

1. Goals of the Course

*"Practical" versus "Foundation" Courses

It is often the case that courses in computer programming are viewed as "vocational" or at least of a practical, or applied nature. Math or English courses are viewed more as "foundations", sources of conceptual skills that can be applied to almost all other mental activities. This hazard is doubly relevant to any proposal for a robotics course.

Why shouldn't introductory programming and robotics courses be regarded as practical or vocational?

We have long maintained (e.g. Aiken and Moshell, 1982; Moshell and Hughes, 1982) that an introductory programming course is primarily an opportunity to develop problem-solving skills and logical thinking. These mental abilities are certainly necessary to the development of good programmers. However, no single introductory course can begin to make an employable computer professional of a total beginner.

To set such goals for a course is to delude both oneself and the students.

We raise the same objection to the view of robotics as a subject matter appropriate for vocational, or perhaps pre-engineering, curricula. Without diminishing a course's usefulness in those roles, it is possible to structure a course that serves much broader goals in developing thinking and reasoning skills.

We would also object to assertions that the teaching of robotics should be motivated by a competitive desire to compete with Japanese (or anyone else's) industrial automation. The goals for our educational system must remain firmly based in general (though "hard-nosed") knowledge and skills. Premature "targetting" of specific technologies is a dangerous form of short-sightedness for educators.

*Course Goals

We intend to develop skill and insight in three areas:

mechanics/physics,
computers,cybernetics,
and planning/problem-solving.

Let's explore each area briefly.

Mechanics

One of the most popular experiments in standard introductory physics is the construction of an electric motor from nails and magnet wire. This motor illustrates rather well the state of electromechnical technology in Edison's time, and teaches a direct, hands-on sense of what magnetism actually is.

We seek the same sort of experience with simple robots. The issues of force, torque, friction and material strengths are fundamental to mechanics; yet most standard lab experience provides very limited intuition into these issues.

The authors' experience with small robots indicates that they are highly motivating, and supply exactly the kind of intuition and practice

with mechanical design that is needed.

The robotics course should have a physics pre- or co-requisite course.

Cybernetics

Programming is usually taught primarily as a hands-on skill. Little opportunity is provided to consider how a program is connected to its users and to the world.

Programs to control robots are conceptually simple, yet their design is difficult. The problem is to understand the delicate relationship between sensory information and control of motion.

This emphasis on feedback, timing, proportional control and error correction constitute a domain called real-time programming. The issues are not intellectually very deep, at the introductory level, yet there is much art in the careful design of working systems.

This presents the course designer with the opportunity to "sneak up" on the student. By presenting stimulating material with important content, in a format that demands many iterations and re-tries, the course can build firm foundation skills in the desired areas.

Organization

For the individual or for a class, the design and construction of a robot is a complex enough task that planning is required.

Merely starting to build something is most unlikely to succeed, because the parts must work together.

A robot project can be factored into several components, with each being handled by a team of students. A typical breakdown is:

-Transporter;
-Hand and Arm;
-Control Devices;
-Computer Software.

The software can be further broken down by tasks, so that one team writes programs to move about the room, another to find something with the arm, another to assemble a structure, etc.

Once one or more operational robots are available, a different level of skill-building is possible. Contests can be held for the most effective programs and strategies to build a castle from blocks; to build a bridge across which the robot walks, etc. When these tasks generate the need to modify the robot, the students have the skills to do so, if they built the robot in the first place. These interactions between task and design are exactly what engineering is all about.

Let us now consider a syllabus that organizes these concepts into a series of lessons.

2. A Syllabus

This syllabus presumes that certain equipment is available; the details of the equipment are described in the following section. Briefly, the following items are needed:

-A simple remote-controlled toy car;

-Components for a three-degree of freedom hand; (an Erector Set with three motors);

-A transporter base; (we use the "Big Trak" toy, and equivalents);

-A microcomputer with a controller capable of sensing the positions of up to 8 microswitches, and of controlling up to 8 motors;

-Some miscellaneous switches, wires and connectors;

-A software system designed for educational robotics.

If the school or individual has an Apple II with 64K of memory and one disk drive, the additional equipment and software needed for this course will cost about $400.

Most of the support software and computer hardware is not yet available for other computers. The tutorial book will include specifications for its construction by dedicated experimenters.

Sequence of Events

Eight lesson modules are intended to span an entire year of high school coursework. The first four modules could be used for a half-year. The timing and relative importance of modules will have to be determined after an experimental teaching of the course.

MODULE 1: INTRODUCTION TO CONTROL

Reading: General concepts of robotics. Robots are general purpose programmable machines which manipulate things. They have senses and use feedback.

Lab: Use remote controlled toy car. Then try to use it when you can't see it; another student gives you instructions and you try to maneuver it through a course. Strategies are analyzed and written down.

MODULE 2: HANDS AND MECHANICS

Reading/Discussion: Degrees of freedom. Force, torque, friction, types of motors (servo, actuator).

Lab: Assemble a simple three-degree arm, controlled by manual "winches". Measure forces on control wires. Test motors with various gearings to see what forces are available.

Use motors to control the arm. Try to pick up small objects.

MODULE 3: SENSORS

Reading/Discussion: Binary (touch) sensors; analog

(angle) sensors. Introduction to programming with these sensors.

Lab: Write a program that displays the status of the arm, using two angular and two touch sensors. Using the direct motor controller, and the program display of arm position, try to pick up an unseen object whose position is known.

Then try to "map" an object by touching it. Is it long or short? How high is it?

MODULE 4: GOAL SEEKING

Reading/Discussion: Feedback, positive and negative.

Lab: Add two microswitch sensors to the remote control car or transporter; one for "edge of table", one for "object in front". Try to find a block of wood on a table without falling off the table, working "blind". Use other students to report car's X,Y position.

MODULE 5: PROGRAMMED GOAL SEEKING

Reading/Discussion: Introduction of program features that control motors, interacting with sensors.

Lab: Try to write a program which does what you just did up in Module 4. First cut: "trivial": car just explores along a one dimensional "track". One direction is fall-off, one is block.

MODULE 6: HAND WITH TRANSPORTER

Reading/Discussion: Balance and Stability; trigonometry for calculation of hand position in space. Sensors for accumulated motor travel.

Lab: Mount the arm on the transporter. Try manual control of the entire system to locate and lift small objects, first visual and then blind (with other students and position sensor program for feedback).

MODULE 7: PROGRAM HANDS

Reading/Discussion: Systematic design of complex programs. How to search a space.

Lab: Program what you did manually in Module 6.

MODULE 8: ROBALL

Reading/Discussion: Description of the game. Two teams, each with one or more robots, attempt to grasp a softball in the center of a pingpong table and, despite all efforts by the opposing team, return it to their own end of the table.

Two divisions: one under manual control, one under computer control.

Lab: Design of a Roball game appropriate to the particular robots available. If only one machine exists, make it a time trial between teams of operators or programmers.

3. Hardware

The primary obstacle to hobby robotics, and the focus of most of the literature, is the design and construction of the mechanical robot itself.

It is rapidly becoming easier to build satisfactory robots for hobby and educational purposes, as more sophisticated components become available. Robotics Age magazine is a good source of inspiration and ideas.

The approach we have taken is to use the most complete subsystems that can be bought, while avoiding the expensive special-purpose modules designed specifically for hobby robotics.

For instance, we use a large Jeep model car from Radio Shack as a transporter base. It is equipped with a proportional servo motor for steering. The "Big Trak" toy is another useful transporter.

In the book we are preparing, the construction of three robots is detailed. The first (R1) is a simple Erector set construction. The second (R2) uses a toy car as its transporter base and has a simple home-built arm. The third (R3, or "Woody"), uses lawn mower tires and a more substantial transporter, and has the most sophisticated arm. All share a common control design and controller unit.

Robot R3 (used in examples later in this paper) has an arm with four degrees of freedom: lateral rotation, whole-arm and forearm elevation, and hand grasp. It has independently controlled left and right wheels, and is supported fore and aft on caster wheels. (See Figure 1)

The following sections describe hardware and software packages being developed to support the tutorial book.

4. Software

Using an unmodified Apple II or Radio Shack computer, it is difficult to control a robot. An interface of some kind is necessary.

Several of the commercially available robot arms use serial ASCII communication, so your computer needs only a serial port such as a printer interface.

The controller under development for this project uses serial communication, but does not require a separate ASCII interface with the Apple II computer. It uses the game controller port and a special software system, instead.

PASPAL for Robotics

PASPAL is a user-friendly Pascal interpreter. It supports advanced animation graphics, and many features designed to help the beginning programmer.

The ROBOT Library Unit for PASPAL adds a variety of data structures and procedures to PASPAL's dialect of Pascal. These commands make it possible to operate the motors of a robot until some

condition on the sensors is met.

The syntax and features described below are preliminary, and are likely to be changed as the system is tested and refined during 1983.

Before we explain PASPAL/ROBOT, we need to briefly discuss motors and sensors.

Motors

The two types of motors in common use in hobby robotics are called "servos" and "actuators".

A servo is a motor whose output is some restricted motion, such as the turning of an arm through 90 or 180 degrees. A servo can be commanded to set its output to some value, such as 50 degrees, and (if it is strong enough to overcome the load imposed on it), it moves to that angle and stops.

An actuator is a motor which can simply be turned on or off. Almost all actuators are reversible. Usually an actuator is paired with some kind of sensor so that the actuator, sensor and controller together behave as a servo. That is, a given angle or position is achieved and the motor is then turned off. Actuators are usually either DC motors, or stepping motors.

If the controller is so designed, a servo can also maintain the position of its output under varying loads. This is important if, for instance, the servo controls the first segment of a multi-segment arm. Usually, however, the friction of the motor and drive train is used to maintain position when the part is not moving.

Small servos are available for $20 to $40 from model airplane shops. Actuator motors are available for prices from a few dollars up, at many hobby shops.

Sensors

Sensors come in many forms, but the three most popular for robotics work are switches, potentiometers and photocells.

A "microswitch" is a very sensitive switch; typically the weight of a sheet of paper will operate a microswitch. These are used to sense a robot's touching something, or the arrival of some part at the limit of its motion.

A potentiometer is a rotary (sometimes straight-line) device whose motion is translated into a varying resistance. The Apple II computer can detect a resistance value between 0 and 150 Kilohms on any of four "game paddle" inputs, so this is a convenient way to report the angle of an arm or shaft.

A photocell is actually a variable resistor, which can be used as a game paddle input or, with a simple circuit, as a switch input to detect the presence or absence of light.

Commands and Functions

The intention of the PASPAL/ROBOT Library Unit is to supply a kind of medium-level command set for our robots.

A low-level command structure would consist solely of commands to start and stop motors, and functions which reported the state of sensors. The difficulties in using such a system are many. Every action must be placed in a REPEAT or WHILE loop, with the program looping until a sensor condition occurs. If the sensor condition was transient, such as the detection of the edge of the table (before falling off), and the loop was slowed by the inclusion of several motor control commands, disaster could occur.

A high-level command structure would contain commands like "GO TO LOCATION 22,45" and "PICK UP A LARGE BALL". Obviously there is a high level of sensor, and program, sophistication required to supply these commands as primitives. It is expected that the creation of these actions as procedures will represent something close to the ultimate capability of the command set to be provided.

A high-level system would also permit several independent actions to occur simultaneously. This degree of parallelism is only possible when the hardware supports reasonably powerful interruption capability. Inexpensive microcomputers such as the Apple II and TRS-80 don't usually have this power, so we are prevented from considering truly concurrent processes in simple command systems.

A concurrent robotics system for more advanced microcomputers such as the IBM and DEC machines will perhaps be developed later. The current system contains some limited concurrent capability, described below in the section titled "BINDING".

We will now describe the "medium level" command set used by PASPAL ROBOT. An example program follows.

Software and Hardware Structures

The software supports up to 32 motor control channels, and 32 sensors (which are usually switches). The hardware is expandable in units of eight motors or sensors up to this limit.

The Apple II also provides four analog inputs.

Sensors may also be defined as "logical sensors" which are combinations of other sensors, or functions of the state of an analog input. In effect, sensors are like boolean functions. The usefulness of this will be apparent later.

The motors and sensors are referred to by integers 0 through 31. CONST declarations in Pascal make these more legible.

We frequently use CONST declarations for the analog sensors attached to game paddle inputs, also.

For instance:

```
CONST  LEFTMOTOR=0
       RIGHTMOTOR=1
       ELBOW=2
       FOREARM=3
       HAND=4
       WHEELS=5
       TOUCHFORWARD=0
       STIFFARM=1
       HOLDING=2
       ELBOWANGLE=0
       ARMANGLE=1
```

These declarations provide us with six symbolic motor names, three symbolic sensor names, and two symbolic analog input names.

In the following, each new command is marked with a star (*) in the left margin, to make it easier to locate.

Sensors

*The boolean function SENSE(which,TOUCH) returns TRUE if sensor number "which" is touching something. Similarly, SENSE(which,NOTOUCH) returns TRUE if the sensor is not touching something. We say that the sensor has value TOUCH or NOTOUCH in these two cases.

The sensors can be combined into "pseudo-sensors" by several commands. For instance, if we execute

*ANDSENSOR(SENSE1,SENSE2,BOTH)

we have now created a new "pseudo-sensor" BOTH (which must be declared with a CONST and a value between 0 and 31, of course.) BOTH will have the value of TOUCH when SENSE1 and SENSE2 both have value TOUCH.

Similarly,

*ORSENSOR(SENSE1,SENSE2,ONE)

would cause ONE to have value TOUCH whenever at least one of SENSE1 and SENSE2 has value TOUCH.

*NOTSENSOR(SENSE1,NOSENSE1)

would cause NOSENSE1 to have value NOTOUCH whenever SENSE1 has value TOUCH.

*ANASENSOR(PADDLE1,GREATER,100,HIGH1)

causes sensor HIGH1 to have value TOUCH whenever the analog input PADDLE1 has a value greater than 100.

*CNTSENSOR(TRIGGER,CNTFLAG,100)

causes sensor CNTFLAG to have value NOTOUCH until TRIGGER has changed to TOUCH and back to NOTOUCH 100 times. Then its value changes to TOUCH.

This is used when we need to do something until a given number of counts are recorded. For instance, TRIGGER might be "toggled" once for each 1/4 revolution of a driving wheel. The sensor counter would be used to measure distance travelled.

The usefulness of these commands will become apparent when we begin to explore the motor control commands.

Motor Control

Motors come in two kinds, servo and actuator. There are three simple "unconditional" commands for the two types of motor.

*SERVO(FOREARM,60)

causes the forearm motor to be set to 60 percent of its fullscale deflection.

*SETDIR(LEFTWHEEL,FORWARD)

causes the actuator motor LEFTWHEEL to run in a forward direction when the motor is activated. Replacing FORWARD with REVERSE would the motor to run backward, whenever it is activated.

*ACT(LEFTWHEEL)

turns on the left wheel. The motor runs for 0.1 second each time this command is encountered in the program.

This simple command would suffice to control some kinds of motion. For instance, if FINGER were a sensor that detected the hand's having closed on something, a loop like

```
REPEAT
 ACT(HANDGRASP)
UNTIL SENSE(FINGER,TOUCH)
```

would cause the hand to close on an object.

However, a smoother and more rapid action can be achieved by combining the ACT and SENSE commands into a "conditional actuate" command:

*CONDACT(HANDGRASP,FINGER,TOUCH)

which has the same effect as the previous REPEAT loop.

Similarly, we have a "conditional servo" command which allows the servo to quit trying to achieve some angle, if a sensor detects a problem.

*CONDSERVO(FOREARM,60,COLLISION,TOUCH)

would cause the servo to try and turn the arm to 60% of its full deflection, unless sensor COLLISION detects a TOUCH.

We also support commands that incorporate counting directly into the motor control.

Assume that DISTANCE is a sensor which "toggles" from TOUCH to NOTOUCH and back again every time the drive wheels rotate 1/4 turn.

*COUNTACT(WHEELS,DISTANCE,40)

would have the same effect as the following:

```
CNTSENSOR(DISTANCE,CNTFLAG,40)
CONDACT(WHEELS,CNTFLAG)
```

That, is the robot rolls forward a distance of ten full revolutions of its drive wheels.

*TIMEACT(WHEELS,20) would cause the robot to roll forward for 2 seconds. (Time is measured in units of 0.1 second).

Several other commands, particularly related to arm positioning and control, are omitted here for reasons of brevity.

Binding

There are times when you want two motors to act in unison; but separate ACT or CONDACT commands would cause one motor to act at a time. For instance, a robot with separate left and right drive motors should use both motors at once to walk forward.

```
*BIND(LEFTWHEEL,RIGHTWHEEL,WHEELS)
```

causes any subsequent actuator commands directed to WHEELS to cause both LEFTWHEEL and RIGHTWHEEL to act.

It is still possible to refer to LEFTWHEEL and RIGHTWHEEL independently. Application of SETDIR to WHEELS would reverse the direction of both wheels when actions were later requested of WHEELS, but not of course when actions were requested of individual wheels.

Example Program

The following program contains several procedures that would be standard parts of any program for R3 ("Woody"). These procedures constitute the "how-to" for retracting the arm, turning in place, and initializing the bindings and counters that are to be used.

The actual main program which uses these procedures is very short. Its purpose is simply to look randomly about a room until it locates a block of wood lying on the floor. R3's arm projects about 4 inches in front of the leading edge of the transporter platform, when the arm is fully retracted. Anything touching the arm is considered to be an obstacle, and activates the sensor OBSTACLE.

A second sensor, on the front caster, detects objects lying on the floor. It is assumed that the wood block is the only thing lying on the floor. This sensor is called LOWTOUCH.

When R3 encounters an obstacle, it backs away and turns a random angle (a multiple of 45 degrees), then resumes walking. This simple a program is capable of being trapped, if the arm is by chance inserted into a hole from which the "back away" maneuver doesn't extricate it. The OBSTACLE sensor would have to be sensitive to touches on both the side and the front of the arm to free R3 from this trap.

Since these commands are imbedded in the PASPAL dialect of Pascal, no semicolons are required. Each command must occupy a separate line.

```
PROGRAM FINDBLOCK
(*
 A Program for Robot R3, in the PASPAL/ROBOT
 Dialect of Pascal.

 Moshell 12/82

 This program guides R3 around the floor randomly
 until a low-lying block of wood is detected.  The
 computer then signals with a "beep" that the block
 has been found.


  SENSORS - - - - - -
*)
 CONST OBSTACLE=1
       LOWTOUCH=2 (* PHYSICAL SENSORS *)

       ANYTHING=3 (* LOGICAL SENSOR *)

(*
  MOTORS - - - - - -
*)
       LEFTWHEEL=1
       RIGHTWHEEL=2 (* TRANSPORTER MOTORS *)

       WHOLEARM=3
       FOREARM=4    (* ARM CONTROL MOTORS. *)
       ARMANGLE=5      (* UNUSED IN THIS PROGRAM *)
       GRIP=6          (* DITTO... *)

       WHEELS=7     (* LOGICAL MOTORS, FOR BINDING *)
       ARMS=8

 PROCEDURE RETRACT (* PULLS THE ARM UP SHORT. *)

 (*
   WHOLEARM retraction is positive (upward); but
   FOREARM retraction requires negative (downward)
   motion.
 *)

 BEGIN
  SETDIR(FOREARM,REVERSE)
  TIMEACT(ARMS,100)
  SETDIR(FOREARM,FORWARD)
 END
 PROCEDURE INIT  (* SETS UP BINDINGS AND ORSENSOR. *)
 BEGIN
  (* FOR ARM RETRACTION : *)
    BIND (WHOLEARM,FOREARM,ARMS)

  (* FOR FORWARD MOTION : *)
    BIND (LEFTWHEEL,RIGHTWHEEL,WHEELS)

  (* FOR DETECTION OF EITHER KIND OF COLLISION : *)
    ORSENSOR (OBSTACLE,LOWTOUCH,ANYTHING)

    RETRACT (* THE ARM *)
 END

 PROCEDURE ROTATE(DEGREES:INTEGER)

  (* Turns the whole robot an angle of
     DEGREES (counterclockwise, looking down,
     is positive.)
```

```
   DEGREES is rounded down to nearest multiple
   of 45 degrees, because that's the accuracy
   of the wheel turning sensor switches.
*)

BEGIN
  IF DEGREES>0 THEN
    SETDIR(RIGHTWHEEL,REVERSE)
  ELSE
    SETDIR(LEFTWHEEL,REVERSE)

  (*
      Sensor ANGTURN pulses once per 45 degrees.

  *)

  COUNTACT(WHEELS,ANGTURN,N DIV 45)

  SETDIR(RIGHTWHEEL,FORWARD)
  SETDIR(LEFTWHEEL,FORWARD)
 END

 PROCEDURE BACKOUT (* Pulls R3 back from obstacle. *)
 BEGIN
  SETDIR(WHEELS, REVERSE)
  TIMEACT(WHEELS,10)
   (* RUN FOR ONE SECOND *)
  SETDIR(WHEELS, FORWARD)
 END

(*
The Main Program
*)

BEGIN

 INIT

 REPEAT

   (* Run forward until you touch anything *)
   COUNTACT(WHEELS,ANYTHING,TOUCH)
   IF SENSE(OBSTACLE) THEN
   BEGIN
    BACKOUT

    (* Rotate 45,90 135 or 180 degrees,randomly. *)
    ROTATE(RAND(4) * 45)
   END

 UNTIL SENSE(LOWTOUCH)

 NOTE(20,200) (* Beep your success ! *)

END.
```

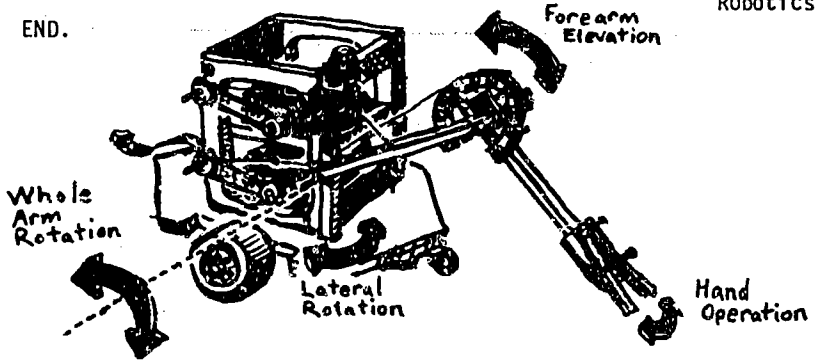


Figure 1: R3 ("Woody"), a Mobile Hobby Robot

Programs for arm control are necessarily somewhat more complicated and require analog arm position sensing.

5. Speculations on Impact

Whereas our earlier effort at designing a programming curriculum ("Computer Power", cf. Aiken and Moshell, 1982 and other papers in bibliography) was explicitly designed for the median student, we do not regard the robotics tutorial as appropriate for that audience.

At the current state of development of hardware and software, the probability of successful, working robot systems is going to be marginal for all but the brightest and most highly motivated students.

This curriculum will, therefore, have to accept the burden of a certain kind of elitism.

Teachers may be surprised, however, at just who the "robot elite" turns out to be. The authors suspect that the skills of "vocational track" students, with tools and materials, may make it possible for them to contribute strongly to the success of such a course.

If the curriculum is successful at teaching technological "basics" such as principles of physical mechanics, it might even turn out to be a subtle path for upward mobility for certain students.

The authors eagerly look forward to in-school tests in academic 1982/3 and 83/4, and will report results in later articles.

Bibliography

Aiken, R. M. and Moshell, J. M. "Computer Power". The Computing Teacher, 9:8, April 1982.

DaCosta, Frank. How to Build your own Working Robot Pet. TAB Books, Blue Ridge Summit, PA. 1979.

Hughes, C. E. and Moshell, J. M. "Rascal and INTERPAS: Graphic Programming Tools for Kids". The Computing Teacher, 9:9, May 1982.

Moshell, J. M. and Hughes, C. E. Robots and the Personal Computer. John Wiley and Sons, to appear in 1984.

Robotics Age. P.O. Box 801, La Canada, CA. 91011.

ENDING THE ISOLATION:
DEAF-BLIND AND MICROCOMPUTERS

by Dan Zuckerman

Human-Interface Laboratory, Science and Technology Studies,
Rensselaer Polytechnic Institute, Troy, NY 12181

Abstract

A method to enable a deaf-blind person to work with a microcomputer is described. Morse Code is used tactilly as a general interface to the screen. Techniques, experiences, and directions for future work are discussed.

Microcomputers can be used to broaden the horizons of the deaf-blind and to greatly expand the number of deaf-blind who are able to interact effectively within the context of sighted hearing people. More than 15,000 people in the United States are both deaf and blind. This overwhelming handicap, being able to neither see nor to hear, was described by Helen Keller as a "dark, silent imprisonment." The lack of educational and training opportunities that are concomitant with their communicative deficits contribute further to the isolation of the deaf-blind. Specialized training is intensive, requires extraordinary committment from the teacher, and requires physical contact for tactile communication.

A minority of deaf-blind, such as Helen Keller, have demonstrated that if the communication barrier can be breached, the deaf-blind can productively and effectively interact with their world. An even smaller minority have found a means and a context in which their handicap is completely eliminated so that they can effectively participate in an international community of seeing-hearing people. These are the deaf-blind who use Morse code and participate in amateur radio. They can communicate by "speaking" with a telegraph key and "hearing" the vibrations from a speaker cone with their fingertips. There is an enormous potential for all deaf-blind people to actively participate in society. By harnessing the power and versatility of today's inexpensive microcomputers with the communication possibilities of Morse code the intellectual and productive potential of the deaf-blind may be released.

My purpose in writing this paper is to describe such a Morse code/microcomputer interface, and the experiences of one deaf-blind user. The interface was implemented on a Radio Shack TRS-80 Model I microcomputer, a machine that costs about $1000. The machine uses the Basic language, has sixteen thousand characters of memory, and uses cassette tapes for long term storage. Development of this software took less than thirty man-hours. The keyboard is used by the deaf-blind individual in the same way most people use it.

The Morse code interface enables a deaf-blind person to interpret the screen just as a seeing person would. As each key is pressed, that character is sounded out in Morse code. These sounds are "heard" by a deaf-blind person as vibrations. Mr. Ray Boduch, the deaf-blind user in this project, developed a circuit to substitute a doorbell buzzer for a speaker so that Morse code can be felt on a sensitive part of the body such as the neck or thigh. An infrequently used key serves as an escape into screen reading mode. The user presses the shift and right-arrow keys followed by a key indicating which line is to be heard. The sixteen lines on the screen are represented by the characters 0 through 9 and A through F. Upon pressing one of these keys, the characters on that line of the screen are communicated to the user in Morse code. At this point, any key will serve to temporarily pause the display and any other key will resume the communication. The enter key is available to stop in mid-line. A utility program is used to control the speed at which information is sent. If the "escape" key is

*I am especially indebted to Dr. Linnda Caporael, friend and teacher, for her help and encouragement with this and other projects, Joseph L. Hartmann Jr., who discovered the Morse code connection and involved me with it, and Ray Boduch, a source of inspiration for myself and many others.

pressed twice, the program will tell (in Morse code) which line on the screen is presently being typed (contains the cursor.)

This software is designed so that it is always available to the user. In general, it permits the use of any software available for the machine and allows the deaf-blind to write their own computer programs. Even if the machine is busy with complex calculations the deaf-blind person can watch the screen as it changes, enabling him or her to share the same perception of the screen as others.

Since Morse code was not designed for computer work, it does not have all the characters needed. Appendix A explains extensions that were developed to display all the characters that a TRS-80 uses. Most of these characters are currently used by ham radio operators. The software interface for this computer is in the public domain and is available from the author.

The interface described above was developed and tested in collaboration with Ray Boduch, who has been blind and deaf almost since birth. By way of background, Mr. Boduch can read Braille, read lips (with his thumb feeling the lips and a forefinger feeling the vibrations in the throat), and speak. At the age of 14, he was taught Morse code and successfully taught the code to a deaf schoolmate. The significance of Morse code is that it eliminates the necessity of tactile physical contact for communication. At 23, with the assistance of a neighbor, Mr. Boduch became a ham radio operator. He can send and receive Morse code at 50 words per minute: the federally set minimum speed competency for an amateur license is 5 words per minute. Presently, Mr. Boduch is employed assembling electronic parts. He hopes to become a computer programmer, an opportunity that is only now possible because of the Morse code/microcomputer interface.

Although it is difficult to generalize from the experiences of one deaf-blind user, the activity is useful for suggesting directions for further research and development, especially for the user side of the human-computer interface. The user side involves both a student and a teacher.

What does an educator of the handicapped need to learn to be able to teach computer skills? To start a person learning computers, a teacher must have a good understanding of a language such as Basic and the techniques for using the available machine. Some Morse code familiarity is also needed to understand the student's feedback. This can be learned with one month of intensive study. Mr. Boduch taught Morse code to a deaf 17-year-old in two weeks. Teachers of computer skills need only have enough background to start their students off. Once the interest is sparked, it is easy to learn independently from information accesssible on diskettes for the system being used. If more than one student is involved, they will learn from each other.

I assume the student knows how to touch-type, has already obtained a grasp of Morse code, and has had the mechanics of starting the computer demonstrated to him or her. The first skills that need to be taught to a new computer user involve reading the screen. The teacher needs to explain that there are sixteen lines of print on the screen, which may change after each keyboard entry. The student needs to be shown how to systematically read the screen by pressing simultaneously the shift and right-arrow keys followed by the number of a line on the screen. The screen should be read from the top down by pressing shift right-arrow, zero to read the first line, shift right-arrow, one to read the second line, shift right-arrow, two to read the third line, and so on. Emphasis should be made on the necessity to methodically read the screen so as to be aware of how it is changing. The user should also be encouraged to read the line he is presently typing. The number of that line may be found by pressing the shift and right-arrow keys twice. The current line number will be heard in Morse code instead of echoing the Morse code signal for the second shift right-arrow. Then, this line may be read by pressing shift right-arrow followed by the character just determined. Characters not previously learned will have to be explained in the context of the computer. For example, the TRS-80 uses ">"' as a prompt to indicate that it is awaiting the user to type something.

Although the deaf-blind effectively use a trial-and-error method for learning, the extreme flexibilty of the computer makes "playing" with it, a common strategy for introducing novices to computing, an unwise choice for the beginning deaf-blind user. It is very important to present simple, clearly shown examples of how program input, listing, and execution differ. The first program should consist of something like "10 PRINT 2+2". Character strings, input statements, and immediate execution mode (a statement with no line number) should not be demonstrated. It is critical at the initial stage that the input program be completely different from the results. Similarity between them will cause confusion. For example, the program 10 PRINT "Hello, I am your TRS-80 Computer." will only confuse the very central issue of how listing a program differs from using a program. The user should also be taught, at least initially, to clear the screen with the CLEAR key before each command. This

will erase the screen without affecting the computer's memory. The user needs to recognize that clearing the screen makes it easier for him to evaluate the effects of a command.

The first multi-step program should be the speed-varying utility. This program uses some statements to control the speed and timing of the Morse output. In order to work with this program, the user needs to change the numeric values contained in the program. The results of doing so and executing the new program will be very apparent: the speed of the Morse code interface will change. Though it is difficult for the deaf-blind user to understand, at this stage, how changing just a few numbers can change the way a computer sends Morse code, the excercise will develop an understanding of just how versatile the machine is. It will also demonstrate how it is possible to defeat the Morse code interface software. This is an important lesson because, as the user becomes more experienced, sophisticated experimentation may possibly do this.

The computer is a very different educational experience for a deaf-blind person because he or she must initiate their learning by using examples, rather than by trial-and-error methods. The deaf-blind do not have the experience of learning by following instructions and understanding examples. Their entire perspective on life is experimental. Very rarely are they presented with instructions for new experiences that make sense to them. In spite of this preference for an experimental approach, it should be emphasized that non-structured experimentation with the computer, rather than trying to understand and repeat the examples presented, will accomplish little. This is a very difficult point to get across to someone whose entire life experience involves tinkering with their environment until the desired effect is achieved.

Presently, Mr. Boduch uses his TRS-80 to exchange letters with a deaf friend via program tapes. The lack of Brailled technical materials has been a shortcoming of using Basic on the machine. The software interface for an IBM personal computer is presently being designed. One advantage of the IBM is that the documentation is available on disk, so it would be accessible with the Morse code interface.

Of course, Braille terminals are available and are used by a growing number of blind programmers. The Versa-Braille is one such device. It has a Braille keyboard and a paperless Braille output device that can mimic the behavior of a video terminal.

Nevertheless, Braille output has an important role. It provides a less immediate method of communication which doesn't require remembering the entire contents of a line. The characters of output can be easily scanned and carefully observed rather than heard just once each time a line is read.

Ray was recently asked what he thought about how readily deaf-blind people could learn to interact with the computer via Morse code. He said that any deaf-blind person who could learn Braille could easily learn this. "Yes, I am very sure, as long as they know what is going on."

A significant number of intelligent, thinking people who are presently blocked by communicative barriers can become participants and contributors to the microcomputer revolution. Technological developments presently available at low cost can be used by educators and others to make a major breakthrough in the welfare of the deaf-blind.

The remaining barriers to this release of intelligence are social, not technological. For example, the Federal Communications Commission requires not only a minimum speed competency in Morse code, but also a technical knowledge of things such as Ohm's law. The requirement of this technical knowledge should be waived for the deaf-blind. Morse code is such a tremendous break-through for the deaf-blind that legislation should be passed to more easily license these people. The world of amateur radio should be made available solely on a statement of desire to communicate via Morse code and a test demonstrating sufficient understanding to communicate via ham radio.

Through the computer "cottage industry", deaf-blind people working at home or in small businesses can compete as equals for employment rather than having no option but to work in sheltered workshops at occupations below their intellectual capacity. No prejudice can affect the person who submits a floppy disk for sale. With the tremendous potential at hand, once a few of the deaf-blind come out into the world, the rest will surely follow.

A row of retractable plastic pins above the Braille keyboard are controlled so as to be readable as Braille. There are at least two advantages of the Morse code/microcomputer interface. One is its low cost and easy access. The other is that the interface is designed to give the user an experience as similar as possible to the usual communication techniques.

Appendix A
Morse Code for the TRS-80

This chart shows the Morse code extensions that have been defined for use with the TRS-80. Both the character (or control code name) and its decimal ASCII representation are given. Appreciation of this information requires a knowledge of Morse code. To find a particular character, combine the Morse sounds of the characters defining that column and row. Here are two examples:

< Column N, Row B, Morse NB
(dah dit dah dit dit dit)

! Column M, Row R, Morse MR
(dah dah dit dah dit)

Blank spaces represent character sounds available for extending this definition to other machines. Spaces that contain boxes are considered sounds too complex for practical use.

	A	N	I	M
A	_ 95	X 88	V 86	Q 81
B	% 37	< 60	SLC *	: 58
C	ESC 27	; 59	ELC **	
D	+ 43	` 96	$ 36	8 56
E	R 82	D 68	S 83	G 71
F	" 34	■■■■■■■■■	■■■■■■■■■	SUB 26
G		SOH 1	& 38	9 57
H		■■■■■■■■■	■■■■■■■■■	■■■■■■■■■
I	L 76	B 66	H 72	Z 90
J		■■■■■■■■■	■■■■■■■■■	
K			HT 9	LF 10
L		■■■■■■■■■	■■■■■■■■■	
M	J 74	Y 89	= 61	NOT ***
N	P 80	C 67	F 70	■■■■■■■■■
O	1 49		2 50	0 48
P		■■■■■■■■■	■■■■■■■■■	{ 123
Q		(40		| 124
R	EM 25	/ 47	■■■■■■■■■	! 33
S	SP 32	6 36	5 53	7 55
T	W 64	K 75	U 85	O 79
U	@ 87	CR 13	4 52	* 42
V		- 45	■■■■■■■■■	} 125
W	[91	■■■■■■■■■	3 51	# 35
X	\ 92	> 62		¬ 126
Y	] 93			DEL 127
Z	94		? 63	■■■■■■■■■

* SLC: Start Lower Case. Upper case is assumed until this character is heard.

** ELC: End Lower Case. This character (as well as CR, ., !, and ?) signals that upper case is now being heard.

*** NOT: Undefined Code. Should a code not in the table be requested, this code is sounded.

In addition to the table, the following characters are defined:

BS	8	IIII
CAN	24	IIIII
US	31	IAA
'	39	AOE
)	41	NQE
,	44	MIM
.	46	AAA

Appendix B
Technical Description of the Morse Code/Microcomputer Interface

This section is provided for the benefit of the technically inclined reader who is interested in implementing the system. The software to accomplish this scheme on a TRS-80 Model I tape-based machine or a TRS-80 Model III disk-based machine has been written. It has been placed in the public domain and is available from the author.

The first implementation was done on a TRS-80 Model 1. It took 475 bytes of code, without provision for upper/lower case. The entire routine resides in the keyboard scan. A tape that patches the keyboard vector and loads it into protected high memory enables the Morse interface.

Upon being invoked, the routine calls the address that was originally in the keyboard vector. The returned character is echoed in Morse code to the speaker port. The routine will return at this point if the key just pressed is not shift right-arrow (ASCII 25). If it is, the keyboard vector is patched to a scan known to be available in the ROM and a character is requested from this keyboard scan.

This character is not accepted until it is a shift right-arrow or a hexadecimal digit representing a line number. If a shift right-arrow, the cursor line is calculated, echoed in Morse code, and the routine returns, after fixing the keyboard vector to point to the Morse interface. Otherwise the hex character is used to calculate a starting address in video memory. The length of this line is determined without trailing spaces. The line is sent in Morse code followed by a carriage return. Before each character is sent, the keyboard is polled with the ROM keyboard scan.

An ENTER will cause the line displaying loop to exit so the remainder of the line will not be heard. Any other key will pause the routine, waiting for a different key to restart the process. The Morse code keyboard routine patches itself back in the keyboard vector before it returns.

The Morse code sending routine uses the cassette port to generate a tone. The most interesting part of this routine is the table encoding the dits and dahs. This table is a bitwise representation of Morse code such that as many as 7 dahs or 14 dits can be represented in two bytes (16 bits). Dits are encoded bitwise as a single 0. Dahs are encoded as a 10. End of character is indicated by a 11. For example, a Morse A (dit dah) is encoded as 0 10 11 and padded with zeroes to form the hex byte 58 (0101 1000).

The character set is encoded in two separate tables. One represents the ASCII codes from 32 to 95. This is used as a look-up table for codes in that range and uses 128 bytes. The second table contains 12 other characters. It uses 36 bytes to encode the ASCII code followed by two bytes of Morse. An ASCII NUL (0) indicates the end of the table. The following two bytes are the character used if the routine is asked to represent an undefined ASCII code.

Thus, it takes 164 bytes to encode 76 characters. The remaining code consumes 311 bytes. The amount of code utilized is a great concern since many TRS-80 applications use almost all of the 16K available on the machine. Implementations on machines with more memory not need be so byte efficient.

PLATO STAYWELL: A Microcomputer-Based Program of Health Behavior Change that Improves With Use

Murray P. Naditch, Ph.D.

Control Data Corporation

PLATO STAYWELL is a microcomputer program of health behavior change. It is highly individualized, matches people to program interventions most likely to be effective for them, and includes branches so that a person's program may be changed if it is not working. The efficacy of person to program matching is evaluated by a mathematical model that enables the program to make increasingly accurate decisions.

Program Rationale

Cardiovascular disease is the leading cause of death in the United States and in other Western industrialized countries. More people die of cardiovascular disease in the United States than all other leading causes combined. Prevention of cardiovascular disease has been a major focus for scientists and clinicians in fields of public health and behavioral medicine because the disease is primarily caused by people's behavior and is to a large degree preventable. A considerable body of research and clinical practice has focused on the development of effective programs of smoking cessation, modification of eating behaviors concerned with dietary cholesterol, salt ingestion, and weight control, modification of behaviors related to more effective control of blood pressure, as well as programs of stress management and physical fitness.

Control Data Corporation initiated a preventative medicine program focusing on cardiovascular behavioral risks in 1979. That program, called STAYWELL, screens employees for potential risks in the areas of smoking cessation, physical fitness, blood pressure mangement, cholesterol and salt consumption, weight control, and stress management. Employees interested in changing health-related behaviors are given opportunities to enroll in courses and other on-site program activities related to health behavior change. This program has been described elsewhere (Naditch, In press).

In 1981, after the STAYWELL program had been implemented in approximately ten American cities, a decision was made to examine the extent to which computer technology could be used to address fundamental unsolved problems in health behavior change.

Research in the area of response to treatment for weight control is illustrative of one of the general problems in this area. In recent years, cognitive behavioral modification techniques have been used with some success in effecting weight behavior change. (For example, see reviews by Stunkard and Mahoney, 1976; Jeffrey, 1976.) Although the results of these studies are statistically significant, the mean group weight losses are often small and do not reach clinical significance. Consistent, significant individual differences among patients in weight loss are observed in studies where individual data are reported (e.g., Harris and Bruner, 1971; Penwick, Filion, Fox and Stunkard, 1971; Gormally, 1979). Consistent individual differences in response to behavior therapies indicate that small mean treatment effects observed in most programs are misleading. This therapeutic approach is effective for some types of patients but not for others.

The importance of identifying individual differences as predictors of success in

behavioral therapies for weight control as well as other behavioral risk areas have been recognized by a number of authors (e.g., Weiss, 1977; Coates, 1977; Leon, 1976). Unfortunately, there has been very little success in finding reliable predictor variables. This lack of success may be due to the fact that most behavior studies: 1) have small sample sizes, 2) use univariate rather than multivariate explanations, and 3) focus on further differentiating the efficacy of program components rather than the interaction of programs and individual responses to those programs.

Another major unsolved problem in the field focuses on continued availability of social support following the end of a formal program. This lack of continued access to social support and/or pervayors of the program after the intervention is over often results in the effects of the program eroding to failure over time.

A final problem that resists solution in the development of effective behavioral interventions concerns the lack of significant cumulative scientific findings in this area. Studies tend to use different definitions of dependent variables, do not all follow up dropout subjects and those who complete the programs for adequate lengths of time, apply varying meanings to the program interventions that they use, and in general lack a sufficiently unified set of definitions or unified scientific paradigm that would be required to produce more cumulative scientific findings.

A Computer-Managed Program Can Address Unsolved Problems in the Field

The PLATO STAYWELL Program has formulated one set of solutions to the problems of individualization, support, and cumulative scientific knowledge. These solutions are based on unique aspects of program users (N = 20,000).

Individualization

The PLATO STAYWELL Program achieves individualization by:

1. Matching people to the programs most likely to be effective for each person,

2. Modifying programs while people are in them as a function of their performance,

3. Tailoring skills to each individuals' needs,

4. Tracking progress and utilizing results to engage in a dialogue-like commentary with each user,

5. Using personal data to interact with each user in a familiar way.

Matching People to Programs

Although there has been a significant body of research that has focused on the effects of individual difference variables and program outcomes, there is very little definitive work that would enable one to effectively match individual differences with the programs most likely to be effective for each person. The PLATO STAYWELL Program has developed a procedure to make this possible. For each behavioral intervention (for example, weight control) the user completes a behavioral profile prior to beginning the program. This behavioral profile contains operationalized versions of the key variables in the clinical literature that have been hypothesized to relate individual differences to program outcomes. For example, in the area of weight control, behavior profile variables include knowledge about nutrition, the degree of social support at home, the degree of overweight, the number of programs the person has been in previously, sex, and other demographic characteristics.

In the initial interation of the program, subjects are randomized across a number of intervention approaches contained within the program. These intervention approaches represent approaches and configurations of program intervention approaches. For example, in the weight control program, some people may either lose weight at the beginning of the program using either a fixed diet, a program of avoiding certain foods and eating others, or a program of calorie counting.

When a sufficient sample of people have run through the program, the individual difference variables are examined using regression equations to determine their efficacy in predicting outcomes at the end of the program and 12 months after the program is over. Individual difference variables that are useful predictors remain in the model and those that do not account for a significant variance are deleted. Variables whose main or interactive effects account for significant variance are then used to match individuals to program paths in the next iteration. This procedure is repeated with each interation, and the system is gradually able to make increasingly accurate predictions about the effects of matching people to program paths.

Program Branching

Each program path includes branches so that individuals who are not doing well may move to an alternative intervention, have the intervention they are in enriched with adjunctive material, or repeat certain aspects of the intervention they have already experienced. Each branch point is treated and tested as an alternative experimental intervention. The efficacy of branch points are evaluated and reconsidered with each new cohort of people comprising one of the iterations in the evaluation process. In this manner, branches may be deleted, new branches may be added, or branches may be kept for people with certain characteristics but not used as branches for people who do not share those characteristics.

Tailoring Skills to Individual Needs

The behavioral profile is supplemented during the program with other self-report data related to an individual's lifestyle and needs. These variables are used to suggest choices or menus of specific lessons to users. For example, in the weight control program, subjects who entertain clients in restaurants, eat many of their meals in restaurants, or who travel frequently are offered lessons focusing on those specific issues. In this manner, people are matched with the skill lessons that are directly relevant to their situation and level of knowledge, and are not exposed to non-relevant lessons.

Tracking and Commentary

The program tracks each user's program history, and uses that data to review progress with each user. Tracking in the weight program, for example, focuses on pounds lost. Tracking in the fitness program, for example, focuses on kilocalories expended, resting pulse rate, and changes in mood since the beginning of the program. This tracked information is presented in a graphic form at the beginning of each lesson. This graphic tracking of progress enables each user to assess their own progress as well as to compare their progress with the progress of other people who have taken the program. As the program data base accumulates information, users will be able to compare themselves with other users who have specific demographic characteristics. For example, a user can ask, "How does my progress compare with that of other white women executives who are my age in this company?"

The commentary and tracking functions are key elements in the user's flow through the program. After a specific program path is selected, the user begins a lesson. Lessons usually involve the introduction of some specific knowledge or skill area relevant to the user, a simulation in which the user is given the opportunity to apply new information in a life-like context, and an assignment through which the individual has the opportunity to try out those specific skills in a real world context prior to the next lesson. For example, in the weight control program, a lesson concerned with eating in restaurants introduces basic skills related to eating and maintaining a low calorie diet in a restaurant, allows the person the opportunity to order a low calorie meal from a simulated restaurant menu in a social context in which other people are strongly and tenaciously encouraging the person to eat a higher calorie meal, allows the individual to track how many calories are in the meal that they have chosen, and makes suggestions for alternative choices and for modes of handling interpersonal situations in which people are encouraging the user to eat more calories than he or she would like.

When the user returns to the next lesson, information is collected about progress and success over the last week. That information is used in the tracking system to present progress, and the user is given an opportunity to engage in a dialogue -like interchange with the computer concerned with the success or failure of the assignment for that week. The dialogue is actually a simple set of options in which the user either moves on to the next lesson, repeats some aspect of the last lesson, modifies their goals into smaller steps, or reconsiders their objectives and moves on to a new course of action.

Personalization Through Familiar Interaction

The computer-managed program is further individualized by having a friendly, supportive tone, referring to the user's name, remembering statements made by the user earlier in the program, allowing users a wide latitude of choice, as well as enabling the tracking of individual's progress and comparison with the progress of other people.

Continued Social Support

The program provides continuing social support by nature of its continued availability. Unlike conventional program interventions, where the instructor or the class disperse when the course is over, the computer terminal continues to be available for the user to come back

to review, continue or reassess progress at any time.

By remembering the user's earlier data, and by having a number of lessons that focus on long-term maintenance, the user is able to experience the program as continuous, friendly, supportive, personal, and responsive even after the user has been away for many months.

The program currently can run either on a free-standing microcomputer using floppy disks or a microprocessor tied into a central computer via a telephone modem. In those instances where there is access to the central computer the program provides a social support network that enables people to communicate with one another about their progress. The primary mode of this communication is through a system called PLATO notesfiles.

Notesfiles are an electronic bulletin board in which users may write statements that can be read by other users. Users reading statements can respond with statements of their own or initiate new statements. Notesfiles can be open to all users or access can be limited to users with specific characteristics. Users who complete the program, or users who are having trouble at various portions within the program related to social support, have the opportunity to read and write in notesfiles of users who are also attempting to stop smoking, lose weight, manage their stress, continue in a program of physical fitness, or manage their blood pressure. Users have the option of creating new notesfiles that may limit group membership, focus on selected topics, or be geographically specific. For example, a localized notesfile can be used to match partners for fitness activities.

A variation of the notesfiles process used in a work site setting involves a program called Action Teams. Action Teams are work site based groups who get together to effect some aspect of the work environment related to more healthy behavior. These groups focus on changing some aspect of the environment such as putting healthier foods in the vending machines or company cafeterias, putting in bike racks or showers, or on some more general health-related activities such as starting an aerobic dance class, having contests among work groups to to lose weight or stop smoking, or initiate intramural sports activities. Each Action Team is led by a lay leader.

The PLATO STAYWELL program recruits Action Team leaders based on their success in the program. People interested in being an Action Team leader take a microprocessor-based lesson on how to be an Action Team leader, and then initiate Action Teams in their work site. A special notesfiles for Action Team leaders is available. Leaders can discuss common problems and have access to expert consultants who can give them advice in initiating and maintaining these groups.

Program Evolution and Theory Construction

The efficacy of the program in producing behavioral change is evaluated using a multiple regression-based structural equation model. This structural equation causal model can be represented pictorially as a causal flow diagram. The flow diagram is one representation of a formal theory, and the model tests the parameters of that theory using regression equations. The theory is redefined with each subsequent iteration of users, and basically serves as a paradigm for the evolution of cumulative scientific research in this area. This scheme is similar to the proposal for the formalization of theory developed by Blalock (1969). In this situation, the evolution and testing of the computer-based instructional model, the clinical model examining the efficacy of person-therapeutic intervention interactions, and the formal theoretical model are synonomous.

Computer Configuration

The PLATO STAYWELL Program runs on a Control Data 110 microprocessor. The program uses two floppy disks that are run on a one-disk drive unit. In each course area, there is a personal disk and a public disk. The personal disk contains information about the indivi-ual's health risk profile, their progress in the program, their history in the program, and evaluation data. The public disk contains specific course lessons.

A program session is initiated when the user inserts his or her private disk. The private disk greets the person, discusses progress during the week, and refers the user to a specific lesson on the public disk. The private disk is removed, the public disk inserted, and the user takes the lesson on the public disk. After completing a lesson on the public disk, the individual returns to the private disk where a specific homework assignment is determined that enables the user to apply the information learned in the lesson.

The program can run either with or without the notesfiles support group and Action Team functions. Notesfiles and Action Team functions require a modem hookup through which the microprocessor can have access to the central computer and to other users.

Programs for weight control, smoking cessation, blood pressure management, and stress management will be in use at Control Data work sites during the first quarter of 1983. Programs for physical fitness and nutrition will be available later in 1983.

Bibliography

Blalock, H. M., Theory Construction: From Verbal to Mathematical Formulation. Englewood Cliffs, NJ: Prentice Hall, 1969.

Coates, T. J., Theory, research and practice in treating obesity: Are they really all the same? Addictive Behaviors, 1977, 2, 95-103.

Cormally, J., Correlates of weight loss and maintenance in a behavioral weight clinic. Paper presented at the Annual Meeting of the American Psychological Association, New York, 1979.

Harris, M. B. and Bruner, C. G., A comparison of a self-control and a contract procedure for weight control. Behavioral Research and Therapy, 1971, 9, 347-354.

Jeffrey, D. B., Behavioral management of obesity: Learning principles and a comprehensive intervention model. In W. E. Craighead, A. E. Kazadin, and M. J. Mahoney (Eds.), Behavioral Modification: Principles and Applications. New York: Houghton Mifflin, 1976.

Leon, G. R., Current directions in the treatment of obesity. Psychological Bulletin, 1976, 83(2), 557-575.

Naditch, M. P., The STAYWELL Program. In Matarazzo, J. D. Miller, N. E. Weise, S. M. Herd, J. A. Weise,S.M. Behavioral Health: A Handbook of Enhancement and Disease Prevention. NY: John Wiley and Sons (In Press).

Penick, S. B., Filion, R., Fox, S. and Stunkard, A. J. Behavioral modification in the treatment of obesity. Psychosomatic Medicine, 1971, 33(1), 49-55.

Stunkard, A. J. and Mahoney, M. J., Behavioral treatment of the eating disorders. In H. Leitenberg (Ed.), Handbook of Behavior Modification and Behavior Therapy. Englewood Cliffs, NJ: Prentice-Hall, 1976.

Weiss, A. R., Characteristics of successful weight reducers: A brief review of predictor variables. Addictive Behaviors, 1977, 2, 193-201.

THE NEUROSCIENCE SOFTWARE PROJECT

by Terry M. Mikiten, Ph.D. and Ronald Pyka

Department of Physiology
and Graduate School of Biomedical Sciences
University of Texas Health Science Center at San Antonio

Abstract

The Neuroscience Software Project was established in 1980 to provide a complete system for delivering computer-assisted instruction in Neuroscience for medical and graduate students. An operating system using Apple II microcomputers was devised for the delivery of instruction and for data collection on student utilization of the system. This report describes the philosophy of the Project, the component parts of the delivery system as viewed by the student user and by the people who manage it, as well as preliminary results gathered after the first year of the system's operation.

Introduction

In order to understand the Neuroscience Software Project (NSP) goals, it will be helpful to briefly describe the course of study it is designed to assist as well as the institutional setting in which it occurs.

Neuroscience is a 98-hour course given to first-year medical students at The University of Texas Medical School at San Antonio, The Medical School is one of several schools in a large Health Science Center. Other institutions include a School of Nursing, School of Dentistry, School of Allied Health Professions and Graduate School of Biomedical Sciences. Because of the subject matter of Neuroscience, the course is taught by people from a variety of disciplines, mostly clinicians and basic scientists from departments in the Medical School and Graduate School. Students taking the Neuroscience Course were from the medical and graduate schools.

The main NSP objective was to provide computer-assisted instruction to the 200 or so students taking the Neuroscience Course. All were linked to what we have called the Primary Assumption of the Project - people using the computers would have no background in either the hardware or software that was involved.

There were a number of other principal objectives in the project. They were as follows:

1. To provide a user-friendly learning environment that emphasized the learning experience, not the hardware.
2. To create a system that was reliable from the standpoint of both software and hardware.
3. To create a system that could be placed in a library setting and could be controlled easily by library personnel.
4. To make both hardware and software available to all individuals who choose to use it.
5. To collect data on the user identity by category, time of use and frequency of use according to individual programs.

The Hardware

The hardware components of the Project consisted of four Apple II microcomputers. Each was coupled to a 12" color television set and a 5 1/4" disk drive. Each Apple II contained 48K of RAM memory, a disk interface card and a CCS clock card. The monitor chip on the motherboard was replaced by a 2716 EPROM containing a modified monitor. This was done to prevent software theft.

This hardware system was mounted in a study carel designed for individual student use. Instructions were mounted on the walls of the carel to remind students of the rules for system operation. Each carel had a powerstrip to which all of the equipment was connected. A switch on the carel wall turned on all the equipment at once. At the same time, power to each carel was controlled by the librarian who activated a numbered switch at the main desk. This system prevented unauthorized use of the systems and gave the librarian an easy way to check which systems were in use. This method forced clients to come to the desk to request that a machine be turned on and so promoted good record-keeping on system use. Figure 1 below shows the overall system diagramatically.

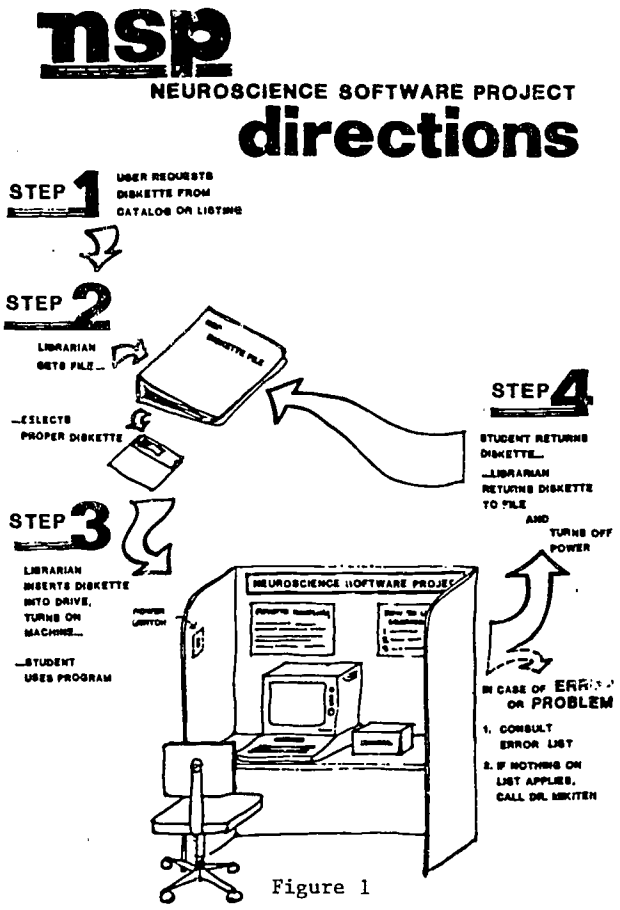


Figure 1

Operation of the System

The Student's Viewpoint: To use software created for the Neuroscience Course, the student requested a topic package of NSP materials from the librarian. The package consisted of both written materials and a diskette. After the student surrendered an ID card, the librarian would switch on one of the carels and give the student the materials. Once the student was at the carel, the sequence of events was as follows:

1. The student placed the diskette in the drive and turned on the local power switch. All programs created for the Project were designed to operate automatically when the system was started in this way.
2. The NSP logo was displayed while the program was loaded.
3. A series of brief menu-oriented questions was run.
4. The main teaching program was run. At the completion of the exercise, the student was asked whether he/she would like to go through the program again. If an affirmative answer was given, the sequence returned to step #3. If a negative response was given, the system ceased operation and power would have to be turned off. To run another program the student would have to start again at step #1.

The Teacher's Viewpoint: From the instructor's viewpoint, the NSP operating system has a variety of features which fulfill the objectives related to security, data collection and ease of management.

Security

Security of the system is maintained at two levels; hardware and software. Both were meant to prevent tampering with either the hardware or programs. Hardware was protected by having padlocks on the back of the Apple II lids. This prevented the machines from being opened. In addition, each computer was fitted with a "reset remover", a small chip inserted in the mother board socket that received the cable from the keyboard. The keyboard cable was in turn connected to the reset remover. This maneuver prevented students from activating the reset key, thereby interrupting the program that was being run.

Software security was achieved by having programs specially coded on the diskettes. The programs were decoded by a machine language program placed into the system monitor on the motherboard. In order to run encrypted programs, the monitor routines were required to convert them to a usable form. Thus it was not possible to take diskettes to non-NSP computers and modify or copy them. Nor could NSP diskettes operate on a non-NSP computer.

Data Collection

In order to determine which programs were used most by the students and who the users were, special efforts were made to collect this information. Each NSP program diskette contained, in addition to the Neuroscience tutorial, an interrogation program which determined the user's identity. Through a series of questions the program determined whether the user was a member of the Health Science Center, then asked if the user was a member of the faculty, student body or other group. If the user was a student, it then asked for the user's School and class level. The program then entered this data, along with the present clock time (read from the hardware clock in the computer) into a random access file on the disk, named the Hostory File. At the end of the program, when the user indicated that it was time to quit, the time was taken once again, and the quitting time was added to the History File.

The History File on a diskette contained the complete history of its use by each individual. Of course, to serve many students, it was necessary to have duplicate diskettes of each NSP program, each with its own History File.

At weekly intervals all of the diskettes were collected and the data from their History Files was read by a Consolidator Program which gathered the data from duplicate diskettes into a Master History File. This file contained the complete

history of use for all NSP programs. Extraction of appropriate data from the Master files was used to determine information about general system utilization, such as total time spent with the NSP programs by all users.

A large number of people used the microcomputers for purposes other than the Neuroscience Course. Because these individuals could not readily be interrogated by an NSP program, data on non-NSP utilization is somewhat less reliable. Since these users had to sign in at the main desk and ask to be assigned to a computer, they were also asked to enter information in a logbook. In addition to name and sign-in time, they were asked to indicate the use to which the computer was to be put. Typical entries were "to play games" or "curiosity". Because the library acquired a significant number of utilities and programming tools, "programming" was also a common entry.

The data gathered from the non-NSP logbook were handled separately from the data collected by the NSP program software. It gave useful information about additional uses to which the computer resources were put and gave a more complete picture of the user community's interests.

Results

System Utilization

The NSP hardware was installed in the Learning Resources section of the Health Science Center Library in the first week of January 1982. Software for the Neuroscience course was made available five weeks later, just prior to the start of the course. Some of the material was related to another ongoing course, Cell Physiology, a prerequisite for the Neuroscience course.

The software remained in place and was available to all students of the Health Science Center for a period of 25 weeks. The total time logged by all four machines over this time was 93,990 minutes. This includes all uses, i.e., NSP and non-NSP. This was accounted for by 1138 individual user visits. Some of these were repeat visits by the same individual; data were not taken to determine the actual number of individuals who used the machines.

Of the total group, 3376 minutes (3.6% of the total) were used for actual use of the NSP programs. The remaining, i.e., non-NSP uses have tentatively been classified into 'game' and 'other' catagories. Game players accounted for 18,264 minutes, or 19.4% of the total non-NSP use.

Game-playing was not discouraged provided that it did not interfere with use of the computers for the NSP programs. For the most part, individuals brought personally-owned games to the computers, although the library did acquire some from a local User's group and made these available. Individuals from all user categories played games. This was not surprising. This aspect of the system's use has increased steadily and by 10-11 months, it has grown to a volume that has elicited some complaints from 'legitimate' users. The complaints most often relate to the noise-level of the games. Other complaints referred to the noise made by groups of children playing games. It was interesting to learn that a large number of game-players were children of the faculty. This was probably the least-expected result of the study. The NSP policy on game-playing is being carefully reconsidered.

System Use Trends

Over the course of this study there was a progressive increase in the system's utilization. Figure 2 below shows total system utilization, expressed as the time spent by all users each week.

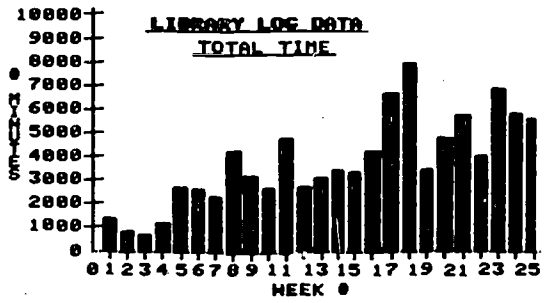


Figure 2

The graph also shows peaks and troughs of use. Closer inspection of the data by NSP program users shows that some of these were linked to Neuroscience-related examinations that occurred in weeks 9, 16, 22 and 26. The examination in week 9 was the final examination in the Cell Physiology Course. Figure 3 gives the data on the NSP programs expressed in minutes. The pattern of use related to examinations is evident here. This graph also shows that NSP use fell dramatically after April 11. This is perhaps not surprising, since the NSP tutorials only covered material presented in the first half of the course.

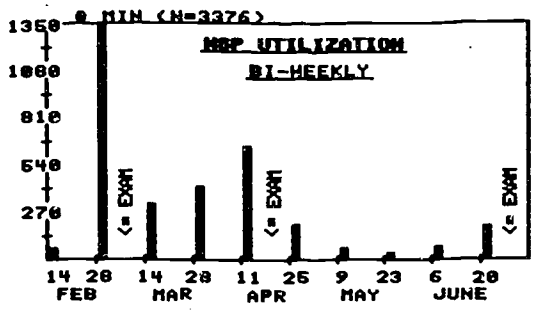


Figure 3

The drop in the utilization around May 23 (week 22) was probably related to something other than examinations in the Neuroscience course. The same drop in utilization was seen in the analysis of off-campus persons who used the programs. Figure 4 shows the pattern of usage for the latter groups of individuals.

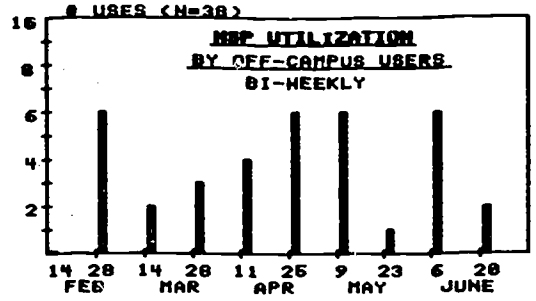


Figure 4

NSP Utilization by Time of Day

Figure 5 below shows the pattern of use of NSP programs during the course of the day. It is evident that, on the average, there was a progressive rise in the number of people who used the system as the day proceeded with a peak around 4:00 p.m. It should be emphasized that this pattern was probably not fixed over the course of the study. More likely, it varied according to the students' academic schedules and the proximity of examinations. The data here show only the cumulative experience.

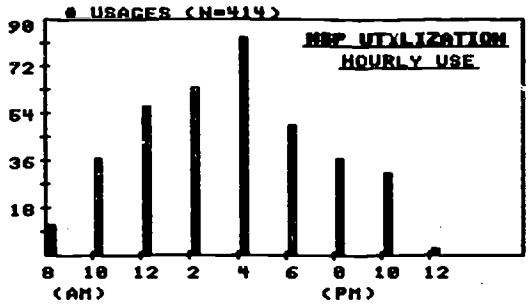


Figure 5

Use According to Student Category

Table 1: NSP Utilization by Group

Group Identity	No. of Individual Visits	Time Spent (min)
Medical Students	244	2295
Graduate Students	42	428
Dental Students		17
Nursing Students	11	43
Faculty	31	131
Miscellaneous	47	303
Off-Campus Visitors	38	159
Totals	414	3376

System Failures

All problems which rendered a carel inoperative were collectively classed as system failures. When a problem occurred, the librarian was asked to report the malfunction by phone. Sometimes a brief consultation with a member of the NSP team clarified the difficulty. On other occasions it was necessary for one of the NSP members to go to the library. On these occasions, the carel was inoperative until the repair could be made. Table 2 below lists the problems that were encountered and the total computer downtime they caused. Data given are approximate values taken from written notes made about each repair.

Table 2: NSP Downtime

Cause	Down Time (min)
TV improperly set by user	30
Damaged diskette	30
Improper insertion of diskette into drive	60
Disconnected video link to TV	30
Oxidized contacts on disk interface card	30
Dislodged Reset Remover	1080
Total	1260

The total system downtime was approximately 1.3% of the total use time. Notice that Reset Removers accounted for a high proportion of the problems we had - 86% of the total downtime. This was evidently due to the propensity of these accessories to dislodge themselves from the 16-pin sockets on the motherboard. The usual repair simply involved reseating the Remover. The time lost because of these devices was partially due to slow response time in answering the initial complaint call for the librarian. In several instances the NSP team was not available and several hours elapsed before a repair could be made. Roughly 10 hours were lost for this reason. If this time (600 minutes) is subtracted from the total downtime, we get a better reflection of downtime related to system malfunction. This corrected value for total downtime then becomes 660 minutes, or 0.7% of the total operating time, a figure that compares favorably with mainframe performance. The problem with the Reset Removers was finally solved by discarding the devices and disabling the Reset key by another means; cutting the Reset line on the Keyboard circuitry.

Conclusions

The system devised for providing computer-assisted instruction operated satisfactorily. Equipment reliability was, for the most part, very high, and the software performed well. The data-gathering system operated extremely well and revealed a few surprising results. Perhaps the most surprising was the overwhelmingly positive user response to the Project. Second was the finding that the utilization for the NSP programs was dwarfed by the other uses to which people put the computers.

Acknowledgements

Several people played important roles in programming the NSP tutorials. They were Shawn Mikiten, Erick Mikiten, and Robert Woodward. John Finley was responsible for the data collection and reduction. Jacqueline Mikiten did the fine artwork and computer graphics that were used throughout. Without the support and help of all of these individuals, the study could not have been done.

We are grateful to Barbara Greene and her staff in the Health Science Center Library's Learning Resource Center for the use of their facility to house the NSP effort. Their kind cooperation and enthusiasm in managing the day-to-day operation of the checkout system was vital to the positive student responses we have received.

COMPUTING IN A NON-CURRICULAR SUPPORT ROLE

J. Spicer Bell
Alonzo D. Peters
Linda L. Royster
Robert W. Jackson
Richard Cornelius
Dr. Randall K. Spoeri

ABSTRACT: A Microcomputer Based Vocational Placement and Follow-up System

J. Spicer Bell, Director, Alonzo D. Peters, Coordinator, Frederick County Board of Education, 115 East Church Street, Frederick, MD 21701

The Vocational Placement and Follow-up service was initiated by the Frederick County Board of Education in the fall of 1981 with a grant from the Maryland State Department of Education under Public Law 94-482. The program serves seven comprehensive high schools and one vocational technical center and is designed to serve a dual function of facilitating initial job placement of graduates and analyzing the success of those graduates on the job. Approximately nine hundred graduates from twenty two different vocational programs register each year and are eligible for employment placement in local businesses. The program provides a systematic design to ensure students equal access to potential job openings and employers a central source of potential employees.

The project was developed around the the use of microcomputer technology to facilitate the handling of large numbers of records and to provide the capacity for quick responses to employer inquiries. Commercially available hardware and software systems were selected because of affordability and the availability of back-up and service. Data collection instruments were designed for use with graduates as they leave the vocational training system and with employers. Student questionnaires provide identifying information as well as information used to screen their eligibility for certain jobs. Employer questionnaires identify potential employment needs and job classifications represented in the employer's work force.

Software systems provide the capacity to rapidly sort graduate files for students with employer identified qualifications. Many times, same day responses to employer inquiries are possible. Current informantion is kept on file through the use of periodic mail and phone follow-up questionnaires. Employer data is kept on a separate data bank and is used by the placement coordinator to periodically canvass the community for available jobs. In addition to immediate information access the system is also designed to provide hard copy back-up in the form of client and employer lists, student referral forms, mailing labels and labor market statistics.

ABSRTACT: Individualized Grade Reports: Motivational Aid and Teaching Tool

Linda L. Royster, Division of Counselor Education, University of Iowa, 338N Lindquist Center, Iowa City, IA 52240

A description of the use and effect of an instructor-made computer grade report, using SCRIPT, a text editing package from the University of Waterloo. The reports were made for junior and senior undergraduate students in a theories of counseling course. The primary pupose of the reports was to provide students individualized statements of their learning accomplishments, and the instructor a structured system of course management. The reports were also used to motivate students, provide diagnostic feedback, demonstrate application of instructional and counseling theory, and validate course performance standards. Rogerian, behavioral, and expectency theory and Bloom's Taxonomy in the Cognitive Domain were used to develop student feedback. Specific student reaction to the reports and class outcomes will be discussed. Copies of reports and the program will be provided.

ABSTRACT: Using a Microcomputer for a Test Question Storage Bank

Robert W. Jackson, 2 Andrews Road, Greenwich, CT 06830

I have repeatedly been asked if a microcomputer could assist the classroom teacher in the preparation of test questions. Inevitably I would reply affirmatively and ask to see the material involved.

Most teachers described a program that would either randomly select questions from a large data base or present the questions for the teacher to select and then print out the test questions neatly on one page and the correct answers on another page. Analyzing the problem I discovered that sequential access was limited by memory constraints of the micros and random access used up too much disk space as it required all files to be as big as the largest question. Normally I would recommend using a commercially available data base manager except that almost all of them limit the material to 255 characters.

I finally came upon a perfect solution to this problem. Most schools have word processor software and almost all word processors allow the material to be saved in ASCII format. If the teacher would use the word processor to enter, edit, and format the questions and answers and then save the large text file in ASCII on the disk, it would be possible to access that text file from BASIC just like a sequential file.

This approach would combine the use of maximum disk space and overcome the 255 limit of random access. Entering and editing the file would be done from the word processor and the selection and printing of the questions would be done from a short BASIC program. I selected the TRS-80 Model III because of its file handling abilities and because the school system that ordered this software used that machine, but the concept can be translated to almost any microcomputer.

ABSTRACT: How Easy to USe Can a Grade Management Program Be?

Richard Cornelius, Wichita State University, Wichita, KS 67208

One way to introduce teachers to computers is to give them a program that saves them time and effort. A grade management program is an obvious choice for this purpose. If a particular program is to be the teacher's first exposure to computers, then every precaution should be taken to be certain that the program is both useful and extremely easy to use. GRADISK is a program that has been written with the overriding goal of making it the easiest to use grade management program available. This presentation will focus on demonstrating how easy to use the GRADISK program is. Features that make it easy to use include:

a) Documentation is complete and includes a sample run.
b) Instructions (to the detail of "press RETURN" where applicable) always appear on the screen.
c) All features are menu-selected.
d) Previously created files are menu-selected.
e) Student records can be examined or edited by identifying a student with a few letters of the last name or the first few digits in a student number.
f) Users are warned before actions that would erase information.
g) Error messages are informative.
h) The program remembers options that you select and streamlines itself for the same selection the next time through.
i) Weighting schemes can be changed at any point during the grading period.
j) The choice of letter grades (e.g., A+, A, A-, ... or pass, fail, or ..) is up to the instructor.

ABSTRACT: An Analysis of Academic Grades at the US Naval Academy, 1971-1981

Dr. Randall K. Spoeri, Major Malcolm W. Fordham, United Sates Naval Academy, Annapolis, MD 21402

In recent years, an area of interest in academic circles has been the phenomenon of grade inflation, or "grade creep". For our purposes, grade creep refers to the steady increase, over time, in grades awarded in academic couses. That is, the inflation of grades over time. This is an area of interest to the Office of the Academic Dean of the US Naval Academy as well.

In consultation with the Academic Dean's office, it was decided to study changes over time of academic grades summarized by:

1) all courses
2) selected courses
3) selected majors
4) all academic departments

Computer data files were established from administrative records stored on computer tapes. The data base was organized to contain all grades awarded to all students for each semester for the academic years 1971 to 1981.

It was decided that the standard quality point ratio (QPR) for academic grades would be suitable for comparing grades for the four types of summary information. FORTRAN 77 programs were prepared to access the data files to develop for each area of interest:

1) the course credit hours
2) a count of A's, B's, C's, D's, and F's by credit hours
3) the percent of each letter grade
4) total quality points
5) total credit hours
6) semester QPR
7) semester standard deviation

The semester QPR's for each semester were placed in data files so that plots of the QPR's over time could be made. Simple least squares linear regressions were applied to each file to provide an indicator of the QPR trend over time.

The study provided useful information to US Naval Academy administrators and department chairmen, as well as showing that there has not been any appreciable grade creep during the period studied.

EXPERIMENTING WITH A COMPUTER LITERACY PROGRAM FOR ELEMENTARY SCHOOL GIFTED AND TALENTED STUDENTS

by W. Starnes and J. Muntner

Montgomery County Public Schools
Rockville, Maryland

Abstract

A computer literacy program for fifth and sixth grade gifted and talented students is described. Computer literacy is a unique vehicle for enhancing the learning experiences of gifted students by providing varied opportunities for differentiation. Because these activities are developed with the characteristics of gifted students in mind, an emphasis is placed on the use of the computer as a tool. Seventy elementary schools participated in the project which centered around three elements: establishing an appropriate learning environment, building creative thinking skills, and planning differentiated computer learning experiences. Ideas and approaches for use by gifted elementary students are suggested.

Introduction

When instituting an innovation in public schools, it is wise to choose a small population on which to try the new idea. Five years ago the Department of Instruction and Program Development in Montgomery County, Maryland decided to purchase a few microcomputers in order to explore how they might be used with gifted and talented fifth and sixth grade elementary school students. The intent was not to restrict computer learning to this population ultimately but to use this group for curriculum experimentation. The school district wanted to determine if elementary pupils could learn beginning programming skills and if they would be interested and motivated to do so. In situations of limited computer access, Seymour Papert advocates looking "for a small pocket of students--perhaps a class of learning disabled youths--and then give them the computers...to show the strength of the computer, which is its ability to help children think."[1] In this case, the choice of gifted students for the program was intentional.

Why Start With Gifted Students?

Computer literacy is a unique vehicle for enhancing the learning experiences of gifted students because it provides many opportunities for differentiation. Classic and current research recognizes that the characteristics of the gifted and talented should determine the nature of their curriculum; in fact, differentiated curriculum must be an outgrowth of those characteristics or learning styles (Kaplan, 1979). The Renzulli-Hartman Behavioral Rating Scale in Chart A lists the learning characteristics of gifted students which are found in the literature.

The task of the educator of the gifted and talented student is to provide experiences and environments which respond to the characteristics of the students by stimulating critical thinking, fostering the use of a scientific approach to problem solving, promoting self-direction and independent work study skills, allowing for use of creative abilities, and providing opportunities for the development of self-evaluation. Activities that stress higher levels of abstraction and concept development are more appropriate for gifted students than those that emphasize rote learning.

The use of the computer as a tool and the development of programming ability are activities that respond to both the identified characteristics of the gifted and talented and the prescribed features of differentiation. In using the computer as a tool students are able to control their own educational environment and build a skill that will be useful in all disciplines.

Implementing the Program

Seventy elementary schools participated in the curricular experimentation with a single microcomputer available to the school for a semester each year. Montgomery County uses the PET microcomputer because these machines are inexpensive, easily portable, and conveniently packaged, having one main body component. They are hardy with an excellent repair rate record which is especially important considering the number of students using the computer in a variety of school environments.

When the computers were first purchased, they were placed in a few schools with some self-instructional materials for student use. Without careful staff orientation, the program often was not implemented.

After that aborted start, the program began with a half-day training workshop for two teachers and the principal from each school. These workshops focused on implementing the program, teaching about computers in society, and hands-on instructional time with the PET. Teachers and administrators worked together to learn their way around the keyboards of the microcomputers. They also

learned some elementary programming and graphics in BASIC. Teachers were encouraged to plan some literacy activities and some initial introduction to programming for class-sized groups to be followed by independent activities for students to pursue.

The goal of the program was to increase the confidence of students in the use and control of computers. The computer literacy activities were designed as a semi-independent instructional unit with objectives in the following categories: understanding computers, working with computers, computers and the society, hardware/software, and attitudes and values.

The chief instructional text used by both students and teachers was the BASIC Manual, written by Montgomery County Public Schools teachers as an interactive approach to teaching beginning BASIC programming. This text uses flowcharts, sample programs and end of chapter exercises to present the content in a constantly reinforcing mode. Supplementing the text were commercially produced games, filmstrips, manuals, magazines, and tapes.

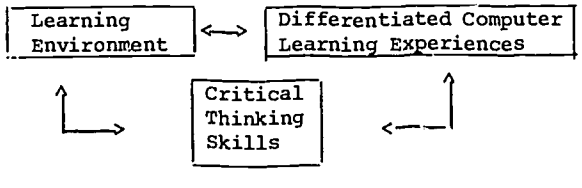
After two years in the pilot situation some elementary schools purchased additional microcomputers and began trying parts of the program with all students. The school district received funding from the Human Resources Research Organization to develop a more generalized K-8 computer curriculum. Elementary teachers who had been exposed to the use of the microcomputer with gifted students began clamoring for additional in-service training.

The official pilot of the project ended last summer. The overall effect of the program was two-fold: to learn that bright fifth and sixth grade students are motivated to and capable of mastering programming skills and to move computer literacy into the elementary schools for all students. A commitment to providing differentiated computer experiences still exists as Montgomery County continues to refine the elements of the Elementary Computer Literacy Program for Gifted and Talented.

What Are The Key Elements?

In Figure 1 is a schematic representation of the elements which were determined by the pilot to be important for implementing a computer literacy program for gifted students.

Figure 1
Elements of the Elementary Computer Literacy Program for Gifted and Talented



Learning Environment

Papert envisions the school in the year 2000 as "a research lab with students engaged in projects and adults function as consultants and counselors."[2] This is a description of the learning environment appropriate for teaching computer literacy to the gifted and talented student. This optimum environment will include the following:

- small groups working together to solve problems of mutual interest
- open-ended tasks
- development of independent ideas
- student involvement and choice in the selection of both resources and projects
- the fostering of discovery learning experiences
- students and teachers working together in interchangeable roles
- self-pacing of activities and projects by students to allow for maximum time for exposure, exploration, and acquiring ownership of concepts
- development and use of self-evaluative techniques by students

The burgeoning appearance of computers in the elementary school classroom has brought with it some feelings of inadequacy and apprehension for the teacher. Teachers have had little experience in either using or teaching about computers. Training courses have been hard pressed to meet the needs of educators. In addition, many students have exhibited remarkable computer skills in very short periods of time. This scenario will only become more prevalent as more and more families purchase home computers. As teachers face the prospect of students with greater knowledge and skills, they will be required to rethink their role as teachers. Nowhere is this more true than in working with the gifted and talented students. Teachers have had to adapt to multi-role concepts as they worked with their young programmers. Molly Watt, writing of her experiences in teaching LOGO, mentions some of the following as appropriate teaching styles: "demonstrator, teacher-lecturer, teller, time structurer, problem setter, management solver, arbitrator, decision maker, challenger, helper, collaborator, process sharer, question asker, idea extender, observer, documenter, admirer, enjoyer, time provider, technician and model learner."[3]

The effective teacher needs to deal with both the cognitive and the affective domains. Traditional teaching roles are not totally abandoned. Students still want and need instruction, but they also have a need for unfettered exploration. The teacher functions as a facilitator, helping students identify and utilize a wide variety of resources. The students' range of materials needs to be broadened as they learn to seek out ideas and solutions in magazines and books and from mentors. An important ingredient in establishing this learning environment is the teacher's ability to admit that he or she does not know all of the answers,

but that she is willing to guide, encourage and appreciate the student's efforts.

While such a learning environment can be established with the availability of only one microcomputer, it is much easier to achieve when several computers are available. PTAs and schools are purchasing additional computers. During the course of this project, summer school courses in computer use were offered using a laboratory setting. The teachers who taught these specialized sessions found that the need for role flexibility was even greater in these situations. In one case, middle school teachers were concerned because they had to establish the summer laboratory with microcomputers from several vendors. They were startled to find that the students leaving the sixth grade had no difficulty moving from one type of computer to another.

Critical Thinking Skills

The second element in the design of the program is an emphasis on the critical thinking skills needed in computer programming. As soon as students begin to practice beginning BASIC, they are asked to predict what an operation will do and to hypothesize about the RUN if the program is changed. Students are challenged to continue to hypothesize as they become more sophisticated programmers. Problem solving is another thinking skill which is emphasized. The logical and inalterable step-by-step nature of learning to program the computer refines the student's problem solving ability. The process of debugging presents a real problem, a program that isn't running as intended. It requires that the student find what the problem is and figure out how to solve it. Very often the solution involves risk-taking, trial and error, and learning from one's mistakes.

The use of creative thinking is an important skill for gifted students. Much emphasis is placed on students creating their own programs. Of course, these programs while new to the students are not unique. The student may be building on some skill he has just learned or refining some previously attempted idea.

The critical thinking skills will not be emphasized if the student is using computer assisted instructional software which emphasizes drill and practice. Games and canned programs can be used occasionally as motivational tools or instructional devices to help students write their own programs.

Differentiated Computer Learning Experiences

Because of their special learning characteristics gifted students can learn faster, acquire in-depth knowledge of an academic area, retain the knowledge, and be expected to use it to create. What relevance does this have for teaching students programming? There are two major ways to ensure that this student has differentiated computer learning experiences: acceleration and horizontal enrichment. Acceleration involves students learning more rapidly and progressing beyond what is expected at a particular grade level. Horizontal enrichment involves a student remaining at a particular skill level but using these skills in a variety of learning activities. These two methods are not discrete, but should be interchanged in different proportions as appropriate for individual students.

What kinds of differentiated computer learning experiences are appropriate for gifted and talented students? The answer to this question must be understood as a tentative answer for the early 1980s when we are just beginning to explore the methods for introducing and working with computers in the elementary school. Several years down the road, so much more will have been learned that present designs will seem inadequate. Present teaching approaches may be viewed as obsolete in the future as the present computers will be. With these disclaimers in mind the following is a list of ideas and approaches useful for gifted and talented upper elementary students.

1. GRAPHICS PROGRAMS. Students can explore a variety of picture and design programs. Using INPUT a program can generate an oriental type rug design with the students name printed in reverse in the middle. Banners and greeting cards are also relatively simple, but satisfying to create. A student created the Happy Mother's Day program reprinted in Figure 2. Sports team banners are particularly successful each season. Also a banner announcing the classroom teacher or class nickname can not only decorate a classroom but assist in class identity. Pictures of hamburgers, people, robots, ice cream sundaes, Snoopy, rockets, Martians and symbols are just a few of the ever popular ideas that students will use to design their own unique graphic programs. Graphics can, of course, be included as an exciting part of all of the other suggested programs, to deliver messages, to provide a title, or to illustrate a particular situation.

2. QUIZZES. These are popular and can be used by other students to enhance a specific study. Elliott, a 6th grader, designed an Egyptian vocabulary quiz entitled Pyramid Power for use in his classroom. The program ended by generating a picture of a pyramid with the user's score on the screen. Students can also be encouraged to contract with another teacher in the building who would like to have a quiz or drill program written for his or her students. In this case, the student has a real simulation of being a programmer and having to design a program to meet the teacher's specifications. Responding to such a request, Stuart designed a program called Planet Weight. The user typed in his or her name and weight. Next a chart was generated on the screen listing the planets of our solar system. The user selected one and

was then told his weight on both earth and the chosen planet. Although Stuart's fifth grade classmates enjoyed this program, it was an even bigger success in the second grade classroom where the students had just finished a study of the universe. The second graders were provided with a beginning computer experience and meaningful learning as well. Stuart received 27 thank you letters showing pictures of him and his computer.

3. MADLIBS. The ever popular game provides the source of another programming idea. Students write their own original Madlib stories and then design the program to ask for the appropriate words and generate a story at the end. Topics are endless from "The Disease" (Figure 3) to "Last Sunday at the Redskin Game."

4. ADVENTURES. This category mixes together adventure, simulation and variations of create-your-own-story. These programs can be written on a very simple level using IF-THEN (see Figure 4) or on a more difficult and exciting one by using RANDOM. Students using the latter BASIC statement will need to graphically represent the program to keep track of its convolutions, and analyze the steps being written. Settings can vary greatly: a wagon train in 1850, a pyramid, or on board a space ship bound for outer space. (Figure 4)

5. ANIMATION. Students moving ahead in their programming knowledge enjoy exploring PEEK and POKE to generate animated pictures on the screen. A birthday cake complete with blinking candles and a happy birthday message greeted one teacher as she entered her classroom on the auspicious day. Moving robots, animals that assemble and disassemble before your eyes, and barren moonscapes that suddenly grow into future cities are other challenging ideas for young programmers.

6. MUSIC. This is another area of exploration for programmers who are no longer novices. Music can be generated for its own sake by budding composers. However, sound and MUSIC can also be incorporated into other programs as mood-setting devices or as a means of delivering a message.

7. GAMES. This is a broad-based catch-all category for homeless program ideas. Word searches based on a theme with a suitable graphic are exciting to design and debug. Scott designed a game called Lucky Sport which incorporated a paper game board. The computer simulated the toss of 2 dice and then asked which of 3 sports symbols the token had landed on. A random event, relevant to the sport was flashed on the screen (ex., you struck out, stay here one turn). Unfortunately, the school year ran out before Scott had a change to figure out how to generate the game board on the computer screen. A thoughtful word is appropriate here. Commercial games and LISTs of game programs have a definite place and value at this level as students strive to create their own and are looking for models of various ways to accomplish specific tasks. The ultimate accolade in this category goes to David, who at age 11 designed a Pac-man type game for the PET computer. (Figure 5) His knowledge far exceeded his teacher's, but she was able to facilitate his work by providing him resources, encouragement and access to a machine.

8. ELEGANCE COMPETITION. This idea can provide a great deal of stimulation and excitement within not only a school, but the community at large. Students are assigned a specific program writing task and compete against each other to create the most "elegant" program to solve the problem. Community members can be utilized as judges. This type of activity requires fairly sophisticated programming ability.

9. PUBLICATIONS. Create a book, newsletter, or a publicity flyer. Use a printer to produce a book of student programs. Sam and Roy borrowed the administration printer to generate the PTA flyers advertising the computer demonstration and discussion. Their flyer included a graphic of a computer and the relevant information. Students have also produced newsletters with a section called DEBUG ME which features programs such as computer riddles, puzzles and cartoons which contain bugs.

The Elementary Computer Literacy Program for Gifted and Talented has been an important component of the overall gifted and talented program in Montgomery County. Feedback from administrators, teachers, parents, and pupils reflect the overwhelming success of the pilot. Eighty-nine percent of the teachers indicated that they would like to continue refining the program and further that they believed the activities were particularly appropriate for gifted and talented students. Eighty-eight percent of the students indicated that they would like to continue in the program in order to further their knowledge of BASIC, write more original programs, and design computer graphics. The staff and students in this pilot would agree with Papert that computers are "carriers of powerful ideas and of the seeds of cultural change...that can help people form new relationships with knowledge that cuts across the traditional lines separating humanities from sciences and knowledge of self from both of these."

References

[1] "Computers Are Objects to Think With", an Instructor Interview with Seymour Papert. Instruction, 3:82, p.85-89.

[2] Ibid., p.87.

[3] Watt, Molly. "What is Logo?", Creative Computing. October, 1982, p.112-129.

[4] Papert, Seymour. Mindstorms, Children, Computers and Powerful Ideas. New York: Basic Books, Inc., 1980, p.25.

[5] Kaplan, Sandra. In-service Training Manual: Activities for Developing Curriculum for the Gifted/Talented. Ventura, California: National State Leadership Training Institute on the Gifted and the Talented, 1979.

Chart A

Scales for Rating the Behavioral Characteristics of Superior Students*

Part I: Learning Characteristics

1. Has unusually advanced vocabulary for age or grade level
2. Possesses a large storehouse of information about a variety of topics
3. Has quick mastery and recall of factual information
4. Has rapid insight into cause-effect relationships
5. Has a ready grasp of underlying principles and can quickly make valid generalizations
6. Is a keen and alert observer
7. Reads a great deal on his own
8. Tries to understand complicated material by separating it into its respective parts

* Renzulli, J.S., Smith, L.H., White, A.J., Callahan, C.M., Hartman, R.K., Creative Learning Press, Inc., 1976.

```
60 PRINT"***GOOD LUCK***"

2 OPEN5,4:CMD5:LIST
10 PRINTTAB(5);"[illegible]"
20 PRINTTAB(5);"[illegible]"
30 PRINTTAB(5);"[illegible]"
40 PRINTTAB(5);"[illegible]"
50 PRINTTAB(5);"[illegible]"
60 PRINTTAB(5);"[illegible]"
70 PRINT:PRINT:PRINT:PRINT
80 PRINTTAB(12);"[illegible]"
90 PRINTTAB(11);"[illegible]"
100 PRINTTAB(11);"[illegible]"
110 PRINTTAB(12);"[illegible]"
120 PRINTTAB(13);"[illegible]"
130 PRINTTAB(14);"[illegible]"
140 PRINTTAB(26);"[illegible]"
150 PRINTTAB(26);"[illegible]"
160 PRINTTAB(27);"[illegible]"
170 PRINTTAB(28);"[illegible]"
180 PRINTTAB(29);"[illegible]"
190 PRINT
200 PRINTTAB(35);"[illegible]"
210 PRINTTAB(36);"[illegible]"
220 PRINTTAB(35);"[illegible]"
230 PRINTTAB(0);"[illegible]"
240 PRINTTAB(0);"[illegible]"
250 PRINTTAB(0);"[illegible]"
260 PRINTTAB(0);"[illegible]"
270 PRINTTAB(0);"[illegible]"
280 PRINTTAB(0);"[illegible]"
290 PRINT
300 PRINTTAB(13);"[illegible]"
310 PRINTTAB(14);"[illegible]"
320 PRINTTAB(14);"[illegible]"
330 PRINTTAB(12);"[illegible]"
340 PRINTTAB(11);"[illegible]"
350 PRINTTAB(11);"[illegible]"
360 PRINTTAB(11);"[illegible]"
370 PRINTTAB(11);"[illegible]"
380 PRINTTAB(12);"[illegible]"
390 PRINTTAB(13);"[illegible]"
400 PRINTTAB(12);"[illegible]"
410 PRINTTAB(11);"[illegible]"
420 PRINTTAB(11);"[illegible]"
430 PRINTTAB(11);"[illegible]"
440 PRINTTAB(12);"[illegible]"
450 PRINT:PRINT:PRINT:PRINT
460 PRINTTAB(11);"[illegible]"
470 PRINTTAB(11);"[illegible]"
480 PRINTTAB(11);"[illegible]"
490 PRINTTAB(11);"[illegible]"
500 PRINTTAB(11);"[illegible]"
510 PRINTTAB(11);"[illegible]"
520 PRINT:PRINT:PRINT:PRINT:PRINT
530 PRINTTAB(16);"LOVE,MIKA"
540 PRINT:PRINT:PRINT:PRINT
550 PRINTTAB(15);"MAY 9,1982"
READY.
```

Figure 2

```
2 OPEN 4,4:CMD4:LIST
70 PRINT"[illegible]"
80 PRINT:PRINT:PRINT:PRINT:PRINT
90 PRINTTAB(6);"**********************************"
100 PRINTTAB(6);"* THE DISEASE-A MADLIBS CRAZE *"
110 PRINT TAB(6);"*                                *"
120 PRINT TAB(6);"*    BY JEFFREY GORDON           *"
125 PRINT TAB(6);"*         AND                    *"
130 PRINT TAB(6);"*       STEVEN SMITH             *"
150 PRINT TAB(6);"*                                *"
160 PRINT TAB(6);"**********************************"
1[illegible]OR T=1 TO 1000
[illegible]XT T
170 PRINT"[illegible]"
180 INPUT"GIVE A NOUN";A$
190 INPUT "GIVE A DISEASE ";B$
200 INPUT"GIVE A NOUN";C$
210 INPUT "GIVE A NOUN";E$
220 INPUT"GIVE A FEELING";D$
230 PRINT"[illegible]"
240 PRINT"ONCE THERE WAS A ";A$
250 PRINT"IT HAD A BAD ";B$
260 PRINTC$;" CAME FROM MILES AROUND TO HELP IT"
270 PRINT"ONE DAY A ";E$;" CAME TO HELP IT"
280 PRINT"IT LIVED ";D$
285 PRINT:PRINT:PRINT:PRINT
290 PRINT "DO YOU WANT TO PLAY AGAIN-Y OR N?"
295 INPUT M$
300 IF M$="Y" GOTO 180
310 END
READY.
```

Figure 3

```
1 C"'H4,4:CMD4:LIST
2 PRINT"J"
3 PRINT"YOU ARE INFRONT OF A PYAMID."
4 PRINT"YOU MUST RETRIEVE A PHAROAHS STAFF,"
5 PRINT"WITHOUT FALLING INTO A TRAP OR BEING CAUGHT BY A CREATURE."
60 PRINT"***GOOD LUCK***"
61 PRINT"YOU ARE IN A DARK HALL IN THE PYRAMID."
62 PRINT" YOU ALMOST FALL IN A DEEP DARK PIT!"
70 PRINT"THEN YOU ENCOUNTER A JACKAL-HEADED"
80 PRINT"GOD."
85 PRINT"SHOOT OR EXIT (S/E)"
90 INPUT A$
100 IF A$ ="S" THEN PRINT"YOU KILLED HIM !":GOTO 120
110 IF A$="E" THEN PRINT"YOU WERE ALMOST KILLED!"
111 PRINT"YOU LEAVE SAFELY ":GOTO 460
120 PRINT "YOU ARE WALKING DOWN THE '.ALL YOU SEE DEEP PIT"
130 PRINT" SHOULD YOU SWING ACROSS OR JUMP ACROSS?(SW/J)";:INPUT B$
140 IF B$="SW" THEN PRINT"YOU BARLEY MADE IT!":GOTO 160
150 IF B$="J"THEN PRINT "!AALLIINNGG!":GOTO 160
160 PRINT"LUCKILY,YOU GRAB AN OLD STONE IN THE HOLE;AND CLIMB UP!"
170 PRINT"YOU ARE WALKING DOWN A HALL ,AND SUDDENLY DARTS FLY AT YOU"
180 PRINT" SHOULD YOU DI' ' TO THE FLOOR OR RUN FOR YOUR LIFE(D/R)?";: INPUT C$
190 IF C$="R"  THEN PRINT"DART OOT YOUR RIGHT ARM!":GOTO 210
200 IF C$="D" THEN PRINT"ALL THE DARTS MISSED YOU!":GOTO 220
210 PRINT"YOU PULLED THE DART OUT OF YOUR ARM AND SUCKED THE POISION OUT"
220 PRINT"YOU SEE A DIM LIGHT ABOUT 200 FEET AHEAD OF YOU."
230 PRINT"YOU RUN TOWARD THE LIGHT!"
240 PRINT"THEN YOU ARE IN THE ROOM AND IS "
250 PRINT"FULL OF TREASURES OF THE PHAROAH'S."
260 PRINT"THE DOOR BEHIND YOU CLOSES!"
270 PRINT"DO YOU WANT TO  STAY OR LEAVE ($/L)";: INPUT O$
290 IF O$="S" THEN PRINT"YOU ARE BRAVE!":GOTO 310
300 IF O$="L" THEN PRINT"YOU MADE IT OUT,CHICKEN!":GOTO 460
310 PRINT"YOU LOOK FOR THE STAFF OF THE PHAROAH"
320 PRINT"YOU ARE READING THE HIEROGLYPHS TO FIND THE STAFF AND A WAY TO GET OU
330 PRINT"YOU FOUND THE STAFF AND A WAY TO GET OUT."
340 PRINT"YOU CAN GET OUT BY:PUTTING THE STAFF IN FRONT OF THE MUMMY CASE."
360 PRINT"TEN CREATURES COME OUT OF THE OUT OF TRAP DOORS IN THE WALL!"
370 PRINT" SHOULD YOU FIGHT OR GET OUT(F/O)";:INPUT E$
380 IF E$="F" THEN PRINT"THE ODDS ARE AGAINST YOU,BUT YOU KILL THEM":GOTO 400
390 IF E$="O" THEN PRINT"THEY GOT YOU,YOU ARE DEAD!":GOTO 460
400 PRINT"A TRAP DOOR OPENS AND YOU WALK OUT INTO THE SUNLIGHT WITH  THE STAFF'
S
410 PRINT" YOU BECOME FAMOUS":GOTO 460
450 PRINT"THEN THE LIGHT ZAPS YOU IN THE FRONT OF THE PYRAMID WITH THE STAFF!"
460 END
READY.
```

Figure 4

```
1 OPEN4,4:CMD4:LIST
5 PRINT"J":DIM V(2)
10 READC,R,P,X,D,S,Q,Y,T,H,J,K
20 DATA20,14,33348,33348,20,12,33268,33268,87,1,3,32
30 DEFFNZ(ZZ)=40*R+C+32768
40 DEFFNW(WW)=40*S+D+32768
50 FORA=33140TO33157:POKEA,160:POKEA+280,160:FORB=1TO15:NEXTB,A
60 FORA=33140TO33420STEP40:POKEA,160:POKEA+17,160:FORB=1TO15:NEXTB,A
70 FORA=33264TO33272:READCC:POKEA,CC:FORB=1TO15:NEXTB,A
80 FORA=33302TO33315:READCC:POKEA,CC:FORB=1TO15:NEXTB,A
90 DATA13,21,14,3,8,32,13,1,14,2,25,32,4,1,22,9,4,32,7,18,15,19,19
100 FORA=1TO1000:NEXTA
101 GOSUB4000
110 FORA=1TO2:PRINT"[redacted]":M=VAL(TI$)+40
120 PRINT"PRESS X TO BEGIN, PLAYER"A
130 GETA$:IFA$<>"X"THEN130
131 TI$="000000"
132 X1$="5":X2$="S":O=0:M=VAL(TI$)+40
135 A$=""
137 P=33348:X=33348:O=33268:V=33268:C=28:R=14:D=20:S=12:U=0
140 GOSUB200
150 GOTO460
200 PRINT"J":X1$="5":X2$="S"
201 POKE32826,81
205 PRINT" SCORE: 0       MUNCH MAN          MEN: 3";
220 PRINT
230 PRINT" [redacted]";
240 PRINT" [redacted]";
250 PRINT" [redacted]";
260 PRINT" [redacted]";
270 PRINT"    [redacted]"
280 PRINT" [redacted]";
290 PRINT" [redacted]";
300 PRINT" [redacted]";
310 PRINT" [redacted]";
320 PRINT" [redacted]";
330 PRINT" [redacted]";
340 PRINT" [redacted]";
350 PRINT" [redacted]";
360 PRINT" [redacted]";
370 PRINT" [redacted]";
380 PRINT" [redacted]";
390 PRINT" [redacted]";
400 PRINT"    [redacted]"
410 PRINT" [redacted]";
420 PRINT" [redacted]";
430 PRINT" [redacted]";
440 PRINT" [redacted]";
450 RETURN
460 TI$="000000"
465 POKEP,81:POKEQ,T
470 GOSUB500
480 GOTO600
500 POKE32826,18:POKE32827,5:POKE32828,1
510 POKE32829,4:POKE32830,25
520 FORAA=1TO1000:NEXTAA:FORAA=32826TO32830:POKEAA,32
530 NEXTAA:RETURN
600 B=B+1:IFB=999999999THENB=0:GOTO600
610 IFB/2<>INT(B/2)THEN1500
620 GETA$
630 IFA$="W"ORA$="A"ORA$="S"ORA$="D"ORA$="X"THENX2$=A$:A$=X1$
640 CY=C:RY=R
```

Figure 5

INTRODUCTORY COMPUTER PROGRAMMING FOR ALL COLLEGE BOUND HIGH SCHOOL STUDENTS

by Ken Jones and Dennis Simms, S.J.

Regis Jesuit High School
Denver, Colorado

Abstract

In this paper we will show an affordable method of developing programming skills for all college bound high school students that does not diminish any of the other skills required of these students.

Many schools cannot afford to use classroom space exclusively for computers. We found a solution to this problem.

In April of 1982, Regis Jesuit High School of Denver, Colorado received a Title 4-C Mini Grant for a project that would develop introductory computer programming as a part of our regular geometry course. Geometry is required of all students at Regis and is taken by most students elsewhere who attend college. This project, therefore deals with the problems of providing computer programming for all college bound students.

THE PROBLEM

1) Regis is a small private school with limited resourses where 98% of the graduates eventually go to college.

2) Many students and teachers at Regis feel that an entire semester of programming is a low priority compared with the other academic demands.

3) The Math Department Chairperson feels that all of our graduates should have hands-on experience with computers and sufficient programming background to allow them to write and understand programs that deal with topics presented in high school.

With these diverse needs and attitudes how can Regis or any other school provide:

A) The equipment and floor space to teach all students programming.

B) The class time necessary to have a satisfactory computer programming experience which will serve as the foundation of computer literacy for these students.

HISTORY OF OUR SOLUTION

In 1978 we built a Digital Group Micro processor. Students helped with the project and they were allowed to use it anytime they wanted. We, at that point, were able to satisfy one student at a time. But it did allow us to get rid of our teletype and the $50.00 a month time-sharing bill.

It was suggested to us by Dr. Ruth Hoffman and her associates at the Denver University Math Lab. that if we could interface a high speed optical card reader with the micro processor we might have an inexpensive solution to our problem. The theory was to set the card reader, the computer and a relatively high speed printer on a cart that you could roll into any classroom. Students would then mark cards and run their programs with almost zero turnaround time.

Students would be able to mark cards at home or anywhere else and we would not have to dedicate a classroom to machines. The cart could be rolled into a closet when a computer class was not in session.

It took almost a year to get everything together, but in the end the system worked perfectly. The computer took no extra floor space, did not require a special room and a class of 25 or more students could get one or two programs running each day. We did indeed have the equipment to reach all students. In addition it was inexpensive, about $3000 for the actual hardware. Cards cost us about $40.00 for 10,000 and it costs about $160.00 to do a semester class, which is well within our budget.

Further, there are educational advantages that exist that should be mentioned.

First, it is not fun to mark cards, therefore students tend to think a little more before they attempt to run a program.

Second, when programs do go wrong, students can work through the program at their desks on the hard copy they have. Once the mistakes are found, corrections can be made and the program can be quickly run again. That is, the cards serve as a storage medium that is quick and reliable.

Our first problem was solved at this point. We had a cheap piece of hardware which a lot of students could use at the same time. But less than one half of our graduating seniors had even taken a computer course. This was not satisfactory.

Would it be possible to take two weeks away from Geometry and teach the whole class how to do area problems on the new computer system?

The answer was a resounding YES! They all learned to operate the machine, mark cards, and to use the Basic commands LET, READ, DATA, PRINT, END and FOR-NEXT. The class' response was, as you might expect, enthusiastic but it was remarkable in several other ways.

First, the students were becoming computer literate. The Micro System has all four basic parts of any computer system. There are:

2 Input Devices - a keyboard and a card reader

2 Output Devices - a CRT and a printer

2 Storages Devices - tape storage and cards

1 C.P.U.

A student can see and touch it all on one cart and with a little help, understand these components as parts of any system.

Second, the students achieved some small power over the machine even in a short period of time. Possibly they could become comfortable with them and not fear them.

Third, there is a link between Geometry and computing. One student went as far as to observe that "programs are much like Proofs in Geometry". We and the National Council of Mathematics teachers agree with him.

SOLUTION TO THE SECOND PROBLEM

How do you provide the class time to have a satisfactory introductory course for all students?

In the results of the experiment indicated above, we could see the possibility of finding a solution to our second problem. The theory is that you can take a substantial amount of time from Geometry to teach programming without hurting Geometry because the two subjects are mutually enhancing.

In January of 1982, we applied for and received a Title 4-C Mini Grant to test our theory.

The grant proposal was divided into two parts. The first part was a pilot program in which four weeks would be taken from Geometry in the Spring of 1982 for one half of our Geometry students. Based on the experience gained with the pilot program the second part of the proposal would develop a six-weeks unit in computer programming and would involve all of our Geometry students in the year 1982-1983.

The main point of this paper will concentrate on the results of the pilot program of 1982. The results of the full program will be presented at the National Educational Computing Conference in the oral presentation of this paper.

During the pilot program we developed a booklet which covered the history of computing, flowcharting, READ, REM, LET, DATA, PRINT, GO TO, IF THEN, END and FOR-NEXT statements and the function SORT. The students are asked to write and run twenty programs using the statements indicated above. These programs are not all related to geometry as they cover some business topics and some topics from algebra.

The objectives for the pilot program included preparation of a syllabus for a modified Geometry course which would include the computer unit and which would not deteriorate the skills of Geometry as measured by our final exam.

We tested this objective in the following way. Three sections or about 90 students took the four-week computer unit. The other three sections did not take the computer unit. They served as our primary control group. This second group spent their extra time on constructions and review of topics that had already been presented.

In May of 1982 the same final exam was given to all geometry students. It was essentially the same exam given in past years. Note, that the previous years results served as a secondary control group. The exam given covered all the material that was common to all sections. It did not cover computers nor was there a heavy emphasis on constructions.

On that test the sections that had taken the computer unit scored better than the control group.

This is exactly what we anticipated. The control group had scored lower on the first, second and third quarter common exams. This is due to slightly lower ability in those classes. However, the difference between the median score for both groups remained the same for all exams. We thought that the difference between the two groups would be smaller on the final exam because of the time taken out for programming. It was not smaller.

When the results of this exam were compared to the results of previous years' exams with the same ability range students taught by the same teacher (our secondary control group), we found that there was less than a 2% difference.

Geometry has not suffered. We have in hand a solution to the second problem mentioned above. It is possible to teach a worthwhile introductory course on computer programming in a Geometry course while at the same time maintaining high standards of academic excellence for Geometry.

The four week period was not long enough to allow all students to finish all of the programs. It was, however, about 75% successful. We anticipate that the six weeks program to be presented this year will take care of that problem. We will be very interested to see how the scores on the final exam for this year compare with those of previous years. These results will be presented in the oral report of this paper.

GEOMETRY SYLLABUS

Time schedule without computers		Time schedule with 30-day computer unit
Chapter 1	THE NATURE OF DEDUCTIVE REASONING	
4 days		4 days
	Emphasis on: If...Ther... Converse, Inverse, Contra-positive, IFF., Only If, Direct Proof, Indirect Proof (Math example only) Some Work on Euler Diagrams	
Chapter 2	FUNDAMENTAL IDEAS: LINES AND ANGLES	
10 days		10 days
	1. The Distance Between Two Points 2. Betweenness of Points 3. Rays and Angles 4. Angle Measurement 5. Complementary & Supplementary Angles 6. Betweenness of Rays	
Chapter 3	SOME BASIC POSTULATES AND THEOREMS	
10 days		10 days
	1. Postulates of Equality 2. The Bisection Theorems 3. Some Angle Relationship Theorems 4. Theorems about Right Angles 5. Some Original Proofs	
Chapter 4	CONGRUENT TRIANGLES	
14 days		14 days
	1. Triangles 2. Congruent Triangles 3. Some Congruence Postulates 4. Proving Triangles Congruent 5. More Congruence Proofs 6. The Isosceles Triangle Theorem 7. Overlapping Triangles	
Chapter 5	DISCUSSION OF REFLECTION AND SYMMETRY ONLY	
3 days		3 days
Chapter 6	INEQUALITIES	
7 days		7 days
	1. Postulates of Inequality 2. The Exterior Angle Theorem 3. Triangle Side and Angle Inequalities 4. The Triangle Inequality Theorem	
Chapter 7	PARALLEL LINES	
9 days		9 days
	1. Parallel Lines 2. Perpendicular Lines 3. The Parallel Postulate 4. Some Consequences of the Parallel Postulate 5. The Angles of a Triangle 6. Two More Ways to Prove Triangles Congruent	
Chapter 8	QUADRILATERALS	
9 days		9 days
	1. Quadrilaterals 2. Parallelograms 3. Quadrilaterals That Are Parallelograms 4. Kites and Rhombuses 5. Rectangles and Squares 6. Trapezoids	
Chapter 9	AREA	
9 days		9 days
	1. Polygonal Regions and Area 2. Squares and Rectangles 3. Parallelograms and Triangles 4. Trapezoids 5. The Pythagorean Theorem 6. Heron's Theorem	
Chapter 10	SIMILARITY	
11 days		11 days
	1. Ratio and Proportions 2. More on Proportion 3. The Side-Splitter Theorem 4. Similar Triangles 5. The A.A. Similarity Theorem 6. Proportional Line Segments 7. The Angle Bisector Theorem 8. Perimeters and Areas of Similar Triangles	
Chapter 11	THE RIGHT TRIANGLE	
13 days		7 days
	1. Proportions in a Right Triangle 2. The Pythagorean Theorem Revisited 3. Isosceles and 30-60 Right Triangles 4. The Tangent Ratio 5. The Sine and Cosine Ratios 6. Trigonometry Is Taught from Unit Circle Approach 7. Use of Hand Held Calculators	
Chapter 12	CIRCLES	
13 days		13 days
	1. Circles, Radii and Chords 2. Tangents 3. Central Angles and Arcs 4. Inscribed Angles 5. Secant Angles 6. Tangent Segments 7. Chord and Secant Segments 8. Inverse Covered by a Discussion	

Chapter 13 CONCURRENCE THEOREMS

5 days — 3 days

1. Concyclic Points
2. Cyclic Quadrilaterals
3. Incircles
4. Ceva's Theorem
5. The Centroid of a Triangle
6. Some Triangle Construction

Chapter 14 REGULAR POLYGONS AND THE CIRCLE

12 days — 7 days

1. Polygons
2. Regular Polygons
3. The Perimeter of a Regular Polygon
4. The Area of a Regular Polygon
5. Limits
6. The Circumference and Area of a Circle
7. Sectors and Arts

Chapter 15 GEOMETRIC SOLIDS

10 days — 5 days

Emphasis on Formulas and Not Proofs

Chapter 16 NON-EUCLIDEAN GEOMETRIES

3 days — 3 days

CONSTRUCTIONS

11 days — 0 days

Changes were made in

Chapter 11, saving 7 days
Chapter 13, saving 2 days
Chapter 14, saving 5 days
Chapter 15, saving 5 days
Construction saving 11 days
30 days TOTAL

The only topics that have been dropped are Ceva's Theorem, Concyclic Points and Limits. All other topics are taken. Less time is spent on topic development and practice as indicated above.

30 DAY COMPUTER UNIT

SYLLABUS

Time	ACTIVITY
3 days	Discuss history, flowcharting and the first program in the text covering REM, READ, LET, DATA and PRINT statements.
5 days	Show how to mark cards and use the machine. Write and run 3 introductory programs. Administer first quiz.
6 days	Discuss Loops, arithmetic symbols and the statements GO TO, IF-THEN and END. Run 3 more programs using these statements. Administer second quiz.
6 days	Discuss counters and debugging. Run 5 programs. Administer third quiz.
6 days	Discuss FOR-NEXT Loops and assign 2 programs using them. Discuss the first 2 programs and assign 3 more. Administer fourth quiz.
4 days	Assign the last 2 programs. Administer final exam on the unit.

30 days TOTAL

A Programming Environment for Preliterate Children

Charles E. Hughes
The University of Central Florida

J. Michael Moshell
The University of Tennessee, Knoxville

Both of
Gentleware Corporation
Knoxville, Tennessee

Abstract

This paper describes a programming environment that provides a gentle, non-textual introduction to programming. The system, named KIDBITS, has been used experimentally for about two years and is appropriate for a wide range of beginners, including children as young as five years old.

Introduction

Many people fail to learn programming because they stumble over the syntax of the language. Remembering facts such as where semicolons go is not an important skill that programming develops; yet, this is the first and, for many, the insurmountable barrier to cross.

When we try to teach programming to young children, the syntax issue is an even greater stumbling block. This is most unfortunate since the real benefit of learning to program comes from the problem solving skills that are developed.

Among the skills developed in learning to translate a problem statement into a programmed solution are the ability to:

Analyse a problem statement, to determine if it is clear enough to allow you to start designing a solution.

Break a large problem into more easily managed subproblems.

Develop an algorithmic approach to solving a clearly specified problem.

Scientifically test a proposed solution to determine if, and under what conditions, it fails to solve the given problem.

Devise a portion of a solution in such a way as to correct this part without polluting other components.

Prepare descriptions of your solutions so that others may understand both what you solved and how you solved it.

Since the activity of programming is so beneficial, it is desirable to introduce it to very young children. Unfortunately, programming environments are, in general, usable only by those who can both read and write. Exceptions to this are certain aspects of the Smalltalk environment (Goldberg 1981, Gould 1982), some implementations of turtle graphics (Papert 1980), programmable toys such as Big Trak, and simple artists tools such as QUILT and PAINTER (Moshell 1982).

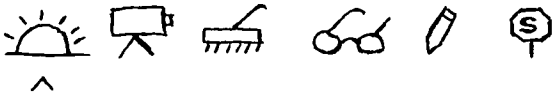
This paper describes a programming environment that provides a gentle, non-textual introduction to programming. The system, named KIDBITS, has been used experimentally for about two years. This is, however the first paper that describes it.

Technical Details

KIDBITS is a programming system in which icons (pictures that represent actions, or objects upon which actions occur) are assembled into sequences to achieve some desired effects. The paradigm used here is one of a movie director (the child) putting together a film. KIDBITS permits the making, editing and playing back of these movies.

Entering a Movie

Typically, the movie's director uses a game paddle to select the components of a movie. The child first sees the following display in the lower part of the screen.



There are six choices here. The first option,"sunrise", means "begin a movie". You use the paddle to position the cursor (^) under the chosen action, and then press the game paddle's button.

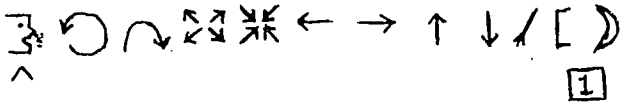
(Cursor control keys can be used on systems having no paddles.)

Here is a description of all the options:

Graphic	Meaning	Details
Sunrise	"BEGIN"	Enter a sequence of commands
Movie Camera	"SHOW"	Replay the movie
Broom	"EDIT"	Modify a movie
Glasses	"READ"	Get an old movie from disk
Pencil	"WRITE"	Save current movie on disk
Stop Sign	"BYE"	End of the day's work

Let's explore what happens when you select BEGIN.

You receive a new menu. Here are its contents.



Graphic	Meaning	Details
Face	"TELL"	Select an animation figure
Counter-clockwise Arrow	"SPIN"	Spin the figure 45 degrees
Clockwise Arrow	"HOP"	Make the figure hop
Four Diverging Arrows	"GROW"	Make the figure grow
Four Converging Arrows	"SHRINK"	Make the figure shrink
Arrows to Left, Right, Up, Down	"LEFT", "RIGHT", "UP", "DOWN	Move the figure
Brush	"PAINT"	Make a copy of the figure
Left Square Bracket	"REPEAT"	Repeat some commands
Moon	"END"	End of the movie

When you select TELL, you receive a new menu of options. This is the list of folks to whom you can "tell" something.

There is a large selection of animation figures available. However, for any given execution, you may have at most six. As an example, we can have a menu that looks like this.

These figures represent:

Flower Bird Cat Rocket Triangle

Here's a typical animation sequence:

```
BEGIN
TELL BIRD       Select the bird
HOP             Bird hops three times to the right
HOP
HOP

TELL CAT        Select the cat
HOP             Cat prepares to pounce on bird
HOP
UP
UP

RIGHT
RIGHT

DOWN
DOWN            Cat lands on bird

TELL BIRD       Select the bird
SHRINK          Bird gets small quickly
SHRINK          (being eaten, perhaps)
SHRINK

END             End taping session for this movie

SHOW            Replay the movie
```

Any number, 1 through 9, may be typed and then becomes the "repeat factor". This number is displayed in the lower right corner of the screen. If you type 3 then select HOP, the current figure hops three times.

Selecting the repeat symbol, a left square bracket, is analogous to using a Pascal or Basic FOR loop: a subsequent sequence of commands is executed some fixed number of times. Prior to choosing this symbol, you enter a repeat factor that determines how many times the following sequence of commands is executed.

As soon as you choose a repeat group, the menu changes slightly. The moon, used to terminate the program, disappears and a right square brackets appears in its place. This icon is selected to close out a repeat group.

An example of the use of the repeat is the following sequence to create a flower garden.

```
BEGIN

TELL FLOWER     Select flower

4
REPEAT          Repeat four times

PAINT           Make a copy of flower
3               Move flower over to make
RIGHT           room for next copy in garden

END REPEAT      End repeated sequence

END
```

Editing a Story

If you have a movie which you have entered into the computer, or loaded from the disk, and you want to change it, select the EDIT option (the broom).

The movie, as currently known to the computer, will be displayed at the top of your screen. This script is read from left to right and from top to bottom, just like English text.

A menu of five choices appears on the bottom of the screen.

The options are:

Graphic	Meaning	Explanation
Right Arrow	Forward	Move forward in the script
Left Arrow	Backward	Move backward in the script
I(nsert)	Insert	Splice new action into movie
D(elete)	Delete	Cut out scene from movie
Q(uit)	Quit	Go back to outer menu

The cursor movement commands (left, right arrow) operate as in a normal scan of text. Moving right of the end of a line positions you at the start of the next one. Moving left of the start of a line positions you at the end of the previous one.

When you select the Insert option, you are given the same menu you used when building the movie in the first place. You use your game paddle to move back and forth and select an item to insert into the movie.

When you select the Delete option, the item pointed to is just removed from the story.

Modes of Presenting this Material

Most children, when learning their native language, master the spoken language by the time they are only three years old. They accomplish this remarkable task by the simple techniques of mimicry and observation of cause and effect.

Several very successful educators have capitalized on this natural mode of learning. The Suzuki method of teaching music does not initially train children to read music. Rather, an adult, usually a parent, is brought in to provide a role model. The child then learns by the familiar methods of mimicry and cause/effect experimentation.

The software we are describing provides an environment for children to enter, modify and play back simple animated stories. Reading is not a prerequisite skill.

The child, with adult assistance, but hopefully not interference, can enter this story via a menu selection scheme. On the bottom of the screen will appear the set of all icons that may be used at the current point of story telling. The game paddle is all that is needed to enter the above story. The position of the paddle controls the cursor that may be moved across the icons.

Pressing the paddle's button enters the currently selected icon as part of the story and shows the effect of this action in the cartoon being generated. Each such selection results in the display of the text of the corresponding Pascal-like statement.

The child who is already reading is lead to learn some of the syntax of a programming language. The child who does not read is helped to develop this skill by the word/action association.

Some of the prominent features of this child appropriate computer story telling system are:

1) Children and adults can work as a team. Adults provide a more highly developed sense of logic plus the reading/writing skills needed for more advanced work. Children provide imagination and enthusiasm. Each learns to program the computer, not be programmed by it.

2) With rare exceptions, e.g., establishing communication with a figure by the TELL verb, each command requires the selection of only one icon.

3) Syntax errors are impossible. Thus the beginner needs to be concerned only about errors of logic. This helps to decrease the frustration level of a high energy, low attention child (or, for that matter, adult).

4) Text is always displayed to correspond with the selected actions. This helps to initiate participants to the next stage of programming.

5) Only one control structure is introduced, the repeated set of commands. This acquaints students with repetitive execution while retaining the essentially imperative nature of the language.

6) Movies may be edited. This early introduction to editing emphasizes debugging and the important realization that you need not throw everything away when your story line is not exactly what you want.

Summary

This paper presents a technical description of KIDBITS' features, and a scenario on how it may be used to introduce children to the art and science of programming. The environment provided by KIDBITS is object-oriented, where the objects are usually cartoon characters. Programming in this system is non-textu... This fact, together with the appealing graph... ...s it very promising for use with preliterate c... .

Certain aspects of KIDBITS have been intentionally designed to reinforce other educational objectives, such as reading and understanding the meaning of single digit numbers.

The present version runs on an Apple II, II+ or IIe computer with 48K of memory and one disk drive. Distribution of the software, along with a self-teaching text, is being done by John Wiley and Sons.

References

Goldberg, Adele and Joan Ross, Is the Smalltalk-80 System for Children?, Byte, Volume 6, August 1981, pp. 348-368.

Gould, Laura and William Finzer, Programming by Rehearsal, Personal Communication, December 1982.

Moshell, J. Michael et al, Computer Power, McGraw-Hill, New York, NY, 1982.

Papert, Seymour, Mindstorms: Children, Computers, and Powerful Ideas, Basic Books, New York, NY, 1981

Teacher Training in Computer Education

William Wagner, Chairperson
Santa Clara County Office of Education
San Jose, CA 95115

ABSTRACT

We are often asked, "How can we ensure that computers will not be just another fad in education, like TV or flexible scheduling?" Since its beginning, the computer education revolution has been different from other innovative episodes because it has featured the involvement and leadership of classroom teachers.

The dilemma for continued successful implementation is to allow each wave of newly involved teacher to experience the sense of ownership, pride and power which is critical in the success of any innovation.

When this need is viewed in the light of the relative lack of technological sophistication of the average teacher, and the lack of new teachers entering the system, the problem of teacher training becomes absolutely essential to the continued progress of this particular innovation at this particular time.

The participants in this panel are four leaders in the field of computer education who have extensive experience meeting these needs in both pre-service and in-service. They bring the perspective of local school district, regional center, state department of education and teacher training institution.

Bobby Goodson
Computer Resource Teacher
Cupertino Union School District
Cupertino, CA 95014

Gary Neights
Bureau of School Improvement
Pennsylvania State Department of Education
Harrisburg, PA 17108

Nancy Roberts
School of Education
Lesley College
Cambridge, MA 02138

Instituting Computer Programs within a School District

John Cheyer
Manchester High School
Manchester, CN 06040

ABSTRACT
This session will describe computer classes or programs being offered by the Manchester School System.

"Principal In-Service Program" - educate principals on different ways the microcomputers may be used in the school system.

"Teacher Awareness Year" - two VIC-20's with packet developed by summer curriculum money explaining hook-up, simple computer programming, and assorted CAI software programs. Each computer stays at each school for 2 months in which teachers can take it home with them to learn at their own pace.

"Elementary Task Force Committee" - will be evaluating pilot programs that are just be introduced in selected elementary schools. For the past two years, we have been evaluating other school district programs which are using LOGO, PILOT, and CAI materials.

"Probe" - two teachers circulate between the ten schools teaching computer programming to the gifted children in the fifth and sixth grades.

"Computer Literacy" - every seventh grader from special education to gifted students receives 15 lessons on computer literacy. Seven lessons are on history, parts of the computer and usage in society and eight lessons are hands on use of the computer.

"Introduction to Microcomputing" - a semester offering to eighth and ninth graders teaching BASIC.

"High School Projects" - The high school has 21 computer available to teach BASIC programming. Another lab is available for CAI. Four departments (English, Mathematics, Science, and Social Studies) have been invited to develop materials using SUPER PILOT or to use courseware from MECC.

Voice Input/Output
New Directions in Instructional Technologies

Carin E. Horn, Chair
Scott Instruments Corporation
Denton, Texas 76201

ABSTRACT

Speech input/output is adjusting the 'traditional' computer assisted learning environment. The panelists in this session will address various aspects of voice recognition and speech synthesis technologies with educational applications.

Remarks will attend to:

a) historical voice I/O developments and applications;

b) the impact of voice I/O on student performance;

c) real-time pronunciation feedback and foreign language learning, and

d) voice recognition for special eduation.

A demonstration of the VBLS (trade mark) voice-based learning system will be given.

PARTICIPANTS

Richard H. Wiggins
Editor, Speech Technology

Herb L. Nickles
California State University

Harry S. Wohlert
Oklahoma State University

Brian L. Scott
Scott Instruments Corporation

Educational Use of Microcomputers by Special Needs Students

Joan Davies, Chair
Lynbrook High School
San Jose, California

ABSTRACT

Just because one happens to work with special education students in Silicon Valley, the heart of California electronics industry, it does not mean that computer instruction automatically becomes part of the curriculum. As much exertion is required here as anywhere else to implement computer use in special education.

Session participants will hear panel members discuss exemplary school site computer programs, as well as, the role of district level and county level personnel, in providing appropriate computer services for special needs students.

Panel members are leaders in the field of compute use for special needs students and have extensive staff development experiences. They will address concerns from school, district, and county level positions.

PARTICIPANTS:

Joan Davies
Lynbrook High School
San Jose, CA

Don Clopper
Coordinator of Special Education
Santa Clara County, CA

Needs and Opportunities for Educational Software in Grades K-12

Edward Esty, Chair
OERI
Washington, DC 20208

Robert Tinker
Technical Education Research Centers, Inc.
Cambridge, MA 02138

Lawrence M. Stolurow
Center for Educational Experimentation, Development and Evaluation
University of Iowa
Iowa City, IA 52242

Darlene Russ-Eft
American Institute for Research
Palo Alto, CA 94302

ABSTRACT

Recognizing the urgent need for improvement in the quality and the quantity of educational software for microcomputer, the National Institute for Education sponsored an eight month study to document the present state of microcomputer use in elementary and secondary schools, and to determine the needs expressed by educators for software which will enable the full potential of microcomputers to be realized.

The results of the survey of teachers, administrators, and software developers will be summarized. The panel will present recommendations for the effective use of microcomputers in the areas of 1) math and science, 2) reading, writing, and communication, and 3) foreign language instruction. Discussion will focus on the ability of microcomputers to improve student motivation and performance, to introduce new topics into the curriculum, to increase teacher productivity, and to decrease educational costs.

The final report of this project should be available at the time of presentation and will contain an exhaustive directory of educational software, commercial and non-commercial sources of software, and a bibliography of pertinent journals, articles, books, and other research studies.

Program Maintenance ... The Forgotten Topic

by Frank W. Connolly

Center for Technology and Administration
The American University
Washington, D. C. 20016

Abstract

Program maintenance is a topic that is frequently overlooked in undergraduate programming classes. This paper offers an approach to presenting the topic. It is based on the author's classroom experience.

Introduction

Programming courses include numerous topics -- structured design, problem solving, language grammar and syntax, interfacing with operating systems, file construction, programming techniques, and more. Discussions with former students brought to light a significant topic I had overlooked for years. A review of several textbooks reflected a similar oversight. The topic? Program maintenance.

Many faculty use a building block approach to teach languages and programming techniques. Students begin with simple programs and then expand them, adding new requirements and using new techniques. Such an iterative approach to program development is a valid teaching approach, but is not what I consider program maintenance. For purposes of this paper, Program Maintenance is defined as modifying, correcting and extending an operating program written by someone else.

Recently, I instituted a learning module on program maintenance in my COBOL classes. That experience is the basis for this paper.

Background

Many students leave the halls of academia to start work at the bottom of the computer career ladder. They are not given programs to create from scratch, as they did in class, nor are they assigned sophisticated new systems to implement. Instead they are given the task of program maintenance. Maintenance tasks are logical assignments for a new programmer. First, as individuals at the lowest rung on the professional ladder, they are assigned work that more senior members of the staff dislike. Second, maintenance tasks are easily defined and controllable, enabling new employees to do productive work without constant supervision. Third, doing maintenance programming gives new staff members the opportunity to learn the accepted norms of their new working environment: programming standards, library contents, and major file formats and conventions, etc.

However, maintenance programming requires knowledge of some special techniques. If students are not introduced to these techniques and requirements, they are ill prepared to effectively handle the initial programming assignments they are likely to receive. Thus, the employer's initial impression of both the new employee and the new employee's education may be nagative. In addition, the psychological impact on neophyte programmers is significant. While adjusting to new work settings, they must contend with programming tasks which are unfamiliar to them.

Environment

The Program Maintenance module I developed was introduced in two introductory COBOL courses. The courses are conducted using strict structured programming techniques. For each programming project, students receive specifications. There are two walkthru's scheduled -- the first at the design stage, the second at clean code. The design phase walkthru requires each student to prepare IPO charts and pseudo-code. The code walkthru requires that the compilation be free of all Warning, Caution and Error messages. There are strict coding style requirements: naming, use of literals, comments, and format. Student teams are formed for each project.

At American University, students use MUSIC (McGill University's System for Interactive Computing). All program code and execution is done on an interactive basis on the University's IBM 4341 via IBM 3270 terminals. From the beginning of the semester, students work in a "library-oriented" environment as program segments and control language are retrieved by students from a public library and incorporated into their code. During the eleventh week of a 15-week semester, the students receive the maintenance assignment.

Assignment

In prior course projects, students received a detailed layout of the input, a description of the processing to be accomplished, and a layout of the output format. For this project, they received:

1. IPO charts of an existing program
2. pseudo-code for the existing program
3. a copy of the existing program source code
4. a description of the modifications required
5. a format of the output for the updated program

After the code walkthru, data and control language for running the updated program was provided.

The program to be updated was written by a former student at the University who understood that the program was to be used by students as described above. It was a short (approximately 150 lines of code), relatively simple program (one file in and a report file out). The code did not adhere to the coding standards used in the course and contained few comments. The IPO and pseudo-code supplied were accurate.

The major restriction placed on the students was that they were not to make unnecessary changes to the existing code. Unnecessary changes were defined as changes made for style or convenience purposes. All new code added was to be in accordance with the style standards established in the class. That meant that the completed code would contain two different styles of writing.

Expectations

While the students appeared to have few expectations about the project, I had several. First, I thought students would enjoy working on a program which had existing code. I anticipated they would find an advantage in being presented a general approach to solve a problem. Having to contend with code not as well-written as they would have it, I hoped to reinforce the need for good structured technique, especially in naming and documentation. I also expected that students would easily make the transition from "creator" of their own programs to "maintainer" of someone else's programs.

Reality

Not surprisingly, reality was significantly different from my expectations. Students did not like doing someone else's programming. They had great difficulty adjusting to the role of "fix-it person." Clearly, the transition from creating to maintaining programs was not as easy as I thought. While the students acknowledged they had learned a great deal, they considered the amount of frustration they experienced as excessive. Upon reflection, I concluded that although the same tools are used for creating and maintaining programs, they are used for different purposes. Therefore, I backtracked and added a new teaching module to explain the maintenance task.

Suggestions and Findings

Finding little material in the literature, I developed the following approach for performing maintenance programming:

I. Determine where you are

What does the existing program do? The analysis begins with IPO charts and pseudo-code, to give the maintenance programmer an overview of the program.

Verify that the IPO and pseudo-code are in agreement both with each other, and the code. It is recommended that the maintenance programmer examine the test data, predicting the output prior to running the program.

Run the program as is, using the test data. If the results agree with the student's predictions, they have an excellent understanding of the code. If they don't agree, they have a road map to indicate where the discrepancies are.

II. Evaluate the changes required

Classify the changes as "Substantive" or "Cosmetic." Cosmetic changes do not affect the logic flow of the existing program. These include such things as headings, layout modifications, field sizes, etc. These generally require changes to the DATA DIVISION, not the PROCEDURE DIVISION.

Substantive changes change the logic flow. These include addition of routines and changes to the sequence of the current program. These changes primarily affect the PROCEDURE DIVISION.

III. Implement the Cosmetic Changes

Without a walkthru, code and then test these changes.

IV. Implement the Substantive Changes

First, prepare IPO and pseudo-code to reflect the program when the substantive changes are implemented. There is a design walkthru at this point.

Second, code and compile the program with all changes included. When it's clean, conduct the code walkthru.

Finally, as with a program written from scratch, test it.

V. Document the program

Document the entire program, not just the modifications.

Student Suggestions

In addition to normal course evaluations, I conducted a debriefing session with students when the project was completed. They had two highly constructive suggestions for using this project in the future.

1. They felt the program I gave them was too easy. As they had written programs of comparable size, they were tempted to rewrite the program, rather than upgrade it. (Several confessed to rewriting it completely. After the output of their totally rewritten program was correct, they went back to their code and replaced portions with the original code where possible, so it conformed to the assignment.) Therefore, they suggested that the original program be larger -- something that would show them the size of a "real world" program. They believed such a program would eliminate the temptation to rewrite the code instead of updating it.

2. They suggested that no new techniques or code concepts be introduced as part of the modifications. The assignment I had presented required them to add control breaks to the existing program. Having worked with accumulators previously, I considered the addition of control breaks a minor new experience. The students felt otherwise. They did not like having their concentration divided between learning control breaks and learning to perform maintenance tasks.

Summary

For 13 years of teaching college-level programming courses, I overlooked a topic of significance to my students -- program maintenance techniques. It wasn't my inspiration or insight that brought it to light. Former students recounting their initial, on-the-job frustration, led me to include a unit on programming maintenance in my introductory programming courses. Based on my review of texts, it is a topic that is forgotten in many courses. But, if we are to prepare students for the world they will face upon graduation, we need to add Program Maintenance to our programming courses.

AN ENVIRONMENT TO DEVELOP AND VALIDATE PROGRAM COMPLEXITY MEASURES

ENRIQUE OVIEDO* AND ANTHONY RALSTON

Department of Computer Science
State University of New York at Buffalo

Abstract

The theory and practice of Software Engineering both require the objective definition and measurement of the complexity of programs. Numerous measures have been proposed for this attribute but little is known about their predictive power and limitations. Extensive empirical studies are needed to determine the usefulness of the proposed measures and to help refine them to the point where they can be used to control and measure effectively the complexity of software products. Programming courses are a useful medium to validate program complexity measures.

(1) Introduction

Program complexity measures vary considerably in the amount of information they require about the program being measured. Some are simple and can be easily automated whereas other measures require human intervention. Obviously, we would like to have automated measures that are easy to use by programmers and that can pinpoint the sources of complexity of programs (and that can give useful and cost effective diagnostics). For the present and for some time there will be no general theory on which to base such measures, therefore, empirical studies are needed to evaluate the advantages of alternative measures, the extent to which they can be automated and the effect on these measures of different programming languages, program sizes and subject domains.

At some stage in the development of Software Engineering we may hope to have a definition of program complexity which is directly derivable from an underlying calculus of programs and programming languages. Pending that time, however, any measure of program complexity needs to be validated by external (i.e., human) perceptions of program complexity. Various experiments have been performed to measure the performance of programmers or to find some common agreement on the complexity of programs. One aspect of the variability of these methods is due to the different assumptions researchers have made about how programmers develop an understanding of programs. We need to explore these experimental methods further to find which ones can provide adequate data on the complexity of programs. In this paper, we argue that college and university programming courses are the best and perhaps the only useful medium in which to carry out experiments to validate and compare program complexity measures and models. Large numbers of programs from many different subjects and in different languages can be obtained from such courses. Moreover, since explicit emphasis on programming methodology is increasingly a feature of even introductory programming courses, a portion of the grade assigned to such projects should be related to program complexity. By having instructors assign a portion of the grade specifically on their evaluation of the complexity of the program, an independent indication of this attribute can be obtained. A data bank of the student programs and their complexity grades can be used to compare alternative measures, refine them and combine them into hierarchical models of program complexity and quality.

(2) Program Complexity: The Nature of the Problem

A basic assumption of this paper is that program complexity is the inverse of program understandability (where we restrict ourselves here to understandability that depends on the structure of the program itself and not on such things as comments or choice of variables names). In order to understand programs people use a combination of the same skills and techniques used for problem solving in general. Induction, deduction, divide and conquer, trial and error and abstraction are a few examples of the mental aids used to develop an understanding of programs [3, 12, 15, 16]. A program can be understood at different levels of detail which range from a general idea of the purpose of the program to a detailed knowledge about the purpose of each control and data structure. The degree of understanding of a program that is achieved and the methods used to obtain it depend on each person's needs, experience with the subject, programming language fluency, etc. The wide variety of programmers' skills and programs' characteristics are the main cause of the difficulty researchers have had in identifying and defining those program attributes which contribute to the complexity of programs.

*The author is currently with the BELL AEROSPACE TEXTRON company, Wheatfield, New York.

Our thesis, and that of several other researchers, is that program complexity needs to be defined as a property of the program itself in order to avoid any subjective components. As a first step in this direction it seems reasonable to assume that the complexity of a program results from a combination of what the program does (i.e., the program's intrinsic functional complexity) and how the program works (i.e., the program's imple - mentation complexity). It would be widely accepted, for example, that a Fast Fourier Transform program is intrinsically more complex than a Bubble Sort program. Similarly, from the implementation complexity point of view, it would be safe to assume that appropriately chosen control structures and data structures make a program more readable and understandable than an unstructured program for the same function [13].

Although the eventual aim of research into program complexity must be to define measures of the implementation complexity of a particular algorithm, in the absence of a general underlying theory we argue that we should be content for the present to define relative measures which can be used to compare a given (e.g., student's) program with a standard (e.g., instructor's) program for the same function[14]. Thus, a plausible initial approach to the study of program complexity would be to analyze measures of program implementation complexity by comparing the corresponding measures of several alternative programs for many different functions. As we learn more about the relative importance of the different program attributes and gain confidence in the existing models and measures, we will hope to achieve gradually the goal of having absolute measures.

(3) Program Complexity: Approaches to Research

Researchers have generally used two different methods to study program complexity. One approach has been to quantify the performance of programmers in order to test hypotheses about the effect of certain program factors on the comprehensibility of programs. The other has been to develop models to compare programs based on their style and organization. Research based on these two approaches has faced serious methodological problems and further research is needed to obtain models that can be used successfully to explain programmer-program interactions [2].

Studies of programmers' behavior have tested hypotheses concerning the effects on program comprehensibility of factors like mnemonic variable names, indentation, paragraphing, alternative control structures and sources of information external to the program such as flow-charts and documentation. In this type of study, the performance of programmers was measured by scoring how well the programmers could memorize, modify, debug, hand trace, answer questionnaires, etc. [8, 9, 10, 15].

Program complexity models have been developed both to achieve a better understanding of the factors that contribute to the complexity of programs and to obtain objective complexity measures. Program complexity models are intended to complement the more developed program correctness models. The amount of knowledge about the language used by the program which is incorporated in program complexity models varies widely. Software Science's model simply distinguishes between operators and operands whereas models based on the program's control flow, data flow or data accessibility require extensive knowledge of the syntax and semantics of the language.

It is reasonable to assume that the more knowledge used in the analysis of programs, the more information can be extracted about the program contents to improve the usefulness of the diagnostics of program complexity models. For example, program complexity models based on the control flow and data flow characteristics of programs could pinpoint anomalies like unreachable sections of code, unnecessary assignments, unfactored expressions, etc., in the program [4,13]. Software Science's model, on the other hand, could only suggest the presence of those anomalies in the program [7]. The use of more information about the program, however, makes more difficult the formulation of simple measures and the validation of the complexity models. This trade-off between the depth of the analysis of the programs required by the different program complexity models and their predictive power is a basic problem in this area which must be resolved [1].

(4) Data Gathering Techniques and Experimental Design

Numerous models and measures of program complexity have been proposed but no major efforts have been made to find out their predictive power and limitations [5]. To test these models and measures we need methods to evaluate independently the complexity of programs. Obviously, these methods have to be designed in relation to the theoretical question being investigated. However, a careful analysis of these methods is essential in this research area because they can suggest how to formulate models and measures that can be more easily studied in an empirical form. We need to determine which (combination of) data gathering techniques provide data which are accurate, unbiased and representative of the complexity of the programs as experienced or perceived by programmers. This is a major task, indeed, given the broad spectrum of programmers' skills, program sizes and functions Some methods which require a direct observation of the behavior of programmers have been used to try to measure the complexity of programs. As noted, one example is quantification of the difficulty (e.g., amount of time) with which programmers could trace, debug, modify, etc., the test programs. A wide variability in the performance of the subjects was found and it would be difficult to develop tests to control or compare the subjects of these experiments based on their skills, training and familiarity with the purpose of the test programs [2, 8]. Although tracing, debugging, etc., are processes that belong to the software development cycle, they do not insure that programmers have understood the test programs or faced all sources of complexity in them. Finally, from a pragmatic point of view, these methods could not be used to gather large amounts of data because they would be too tedious

and time consuming with non-trivial test programs.

We believe more comprehensive and adequate data can be obtained from experiments where properly trained (panels of) experts judge the relative complexity of alternative programs for the same function. A general argument in favor of this contention is that reading and judging programs is an essentially simpler process than writing, tracing, modifying, etc., them thereby making easier the collection of data. Reliable data may be expected from expert judges who have a clear understanding of the purpose of the programs, the language used and the techniques used in their implementation (e.g., abstract data types, recursion, etc.). Additionally, a relatively large number of alternative implementations should help the judges give a well informed opinion of the relative complexity of the programs.

Program complexity is an elusive and multifaceted concept. Therefore, extreme care must be taken in the design of case studies and experiments. For example, comprehensive and unbiased guidelines should be given to the experts judging the complexity of programs. These guidelines should enhance the panelists' understanding of the concept of program complexity, carefully define the program attributes they should (not) consider in the evaluation of the programs, how finely they should classify the programs according to their complexities, etc. Similarly, test programs have to be carefully selected in order to control those program attributes not taken into account by the particular model or hypotheses being tested (i.e., insure the internal validity of the experiment) and to test the models or hypotheses with a variety of program types, sizes, and programming languages (i.e., insure the external validity of the experiment).

(5) An Exemplary Experiment to Test a Model of Program Complexity

We have developed a model of program complexity which is based on the control flow and data flow characteristics of programs (see below). The case study we present here is part of an ongoing effort to test this model. The test programs for this experiment were written by students of an introductory computer science course and a preliminary analysis of the behavior of the model is done by comparing the complexities assigned to these programs by the model and the graders of the course. We do not regard the results of this experiment as a definitive test of the model but as possible insights into the data gathering technique discussed in this paper and the behavior of the model that should be further explored.

We first give a brief description of the model which is sufficient to understand the data gathering method we have developed to validate it. A detailed definition and analysis of the model can be found in [13].

Control structures and data structures are fundamental and closely intertwined program components (as exemplified by the title of Wirth's book [18], "Algorithms + Data Structures = Programs"). It would be widely agreed that the appropriate choice of control and data structures can make programs more readable, easier to debug and more reliable. This general argument and the results of studies that suggest that the control flow and data flow of programs play an important role in the ability of programmers to understand, trace and debug programs [4, 6, 9, 11, 12] have led us to formulate a model of program complexity based on the control flow and data flow characteristics of programs.

The model of program complexity we have defined is based on the assumption that, in order to understand, trace and debug a program, a programmer must, at least, be able to determine a) the statements that precede and the statements that follow each statement in a program, and b) the set of variable references affected by each variable assignment and the set of assignments by which each referenced variable could have been defined.

Based on this assumption we have defined two program attributes called control flow complexity (CF) and data flow complexity (DF). These attributes (CF and DF) have been defined in a language independent form and we have implemented a system that measures automatically the CF and DF of Pascal programs. In order to define control flow complexity, we view a computer program as a flow graph having a single entry and a single exit node and a number of nodes in between where each node represents a sequence of statements without branches into it or out of it and the nodes are connected by edges that represent paths through which the dynamic execution of the program flows. We assume that the difficulty in understanding the sequences of node executions (i.e., its control flow complexity CF) can be measured directly by the number of edges of the program flow graph.

Several studies indicate that the size of the set of variable references affected by each variable assignment and the set of assignments where each referenced variable could have been defined have a significant effect on the ability of a programmer to understand, trace and debug a program. We surmise that the difficulty in understanding the definition-reference relationships in a program (i.e., its data flow complexity DF) can be quantified by counting the number of variable definitions (references) associated with each variable reference (definition) in that program.

The fifty-nine programs used in this experiment were the first programming assignment of an introductory computer science course. Only correct programs were used in order that the assessments by graders of the implementation complexity not be affected by syntactic or semantic errors in the programs. These programs were written in Pascal and they all compute the cube root of a set of numbers provided in an input file. Their output was a list of the numbers read from the input file and the cube root corresponding to each number. All programs consisted of a single main program with no procedures or functions.

Two computer science graduate students

assessed the complexity of the programs as excellent (A), good (B) or poor (C) (with "excellent" corresponding to "less complex"). One student graded 33 programs and the other graded 26 different programs. The graders were told only to judge complexity in terms of the choice and use of control structures and data structures in the programs but they were not told what the details of the model being tested were. In order to enhance the graders' understanding of the concept of program complexity, they were asked to consider the set of questions concerning some general and specific program characteristics shown in Figure 1. It was our intent that these questions not give clues about the model being tested. Additionally, to ensure a consistent classification of the programs, we instructed the graders to scan all the programs before grading any of them to form a clear notion of what constituted an excellent, good and poor implementation, and to ignore program attributes not considered in the experiment such as mnemonic variable names, indentation, comments, etc.

DATA FLOW (DF) RELATED QUESTIONS

1. Are the data structures appropriately chosen?
2. Are the data structures properly used?
3. Are the boolean conditions properly used?
4. Are there unnecessary variables?
5. Are there duplicated sections of code?
6. Are there variables that should have been declared as constants?

CONTROL FLOW (CF) RELATED QUESTIONS

1. Are control structures appropriately chosen?
2. Is the structure of the algorithm reflected in the control structures used in the program?
3. Can nested if's be comprised in simpler conditional statements, as
   if b1 then if b2 then X = A versus
   if b1 and b2 then X = A?
4. Is the level of nesting of control structures not so deep as to make the program unintelligible?

CF AND DF RELATED QUESTIONS

1. Is it easy to find and understand the conditions under which a section of code would be executed?
2. Is it easy to follow the sequence of statements in each path of the program ?
3. Is it necessary to backtrack often in order to follow the implementation of the algorithm?
4. Could certain sections of the program be programmed in a simpler way?

FIGURE 1

We have used the data obtained from this experiment to compare how well the control flow complexity (CF), data flow complexity (DF), cyclomatic complexity (V) [11] and a combination of control flow complexity (CF) and data flow complexity (DF) correlated with the complexity grades assigned by the graders. The grader of the 33 programs assigned 13 A's, 15 B's and 5 C's. The grader of the 26 programs assigned 4 A's, 16 B's and 6 C's. In Figures 2 and 4 we have plotted the control flow complexity (CF), data flow complexity (DF) and complexity grades (A,B,C) assigned by the graders corresponding to each test program for the 33-set and 26-set, respectively. Figures 3 and 5 show tallies of the complexity grades assigned to the programs and their corresponding cyclomatic complexity (V) and control flow complexity (CF) for the two sets, respectively.

In order to compare roughly these measures of program complexity, we have defined areas for A, B and C-programs in Figures 2-5 and then have counted how many programs have appropriately and inappropriately fallen into each area. Our criterion to obtain these areas has been to take the maximum V, CF and DF of the A, B and C-programs as the limits for the maximum complexities that A, B and C-programs can have. The maxima for V, CF and DF for the programs from the two sets are shown in Tables 1 and 2, respectively. The zones defined by these limits are shown as dashed lines in Figures 2-5.

TABLE 1

	Max. V	Max. CF	Max. DF
A-programs	4	13	10
B-programs	6	19	16
C-programs	7	22	28

TABLE 2

	Max. V	Max. CF	Max. DF
A-programs	4	13	9
B-programs	5	16	17
C-programs	7	30	29

Tables 3 and 4 show the number of A, B and C-programs contained in the A, B and C-zones of Figures 2 and 3, and 4 and 5, respectively. Based on the information provided by Tables 3 and 4, we might conjecture that a combination of CF and DF can classify the programs better than V, CF or DF alone, but obviously this hypothesis remains to be proved. We emphasize that this case study was presented solely to illustrate a method for gathering data and validating models of program complexity.

FIG. 2

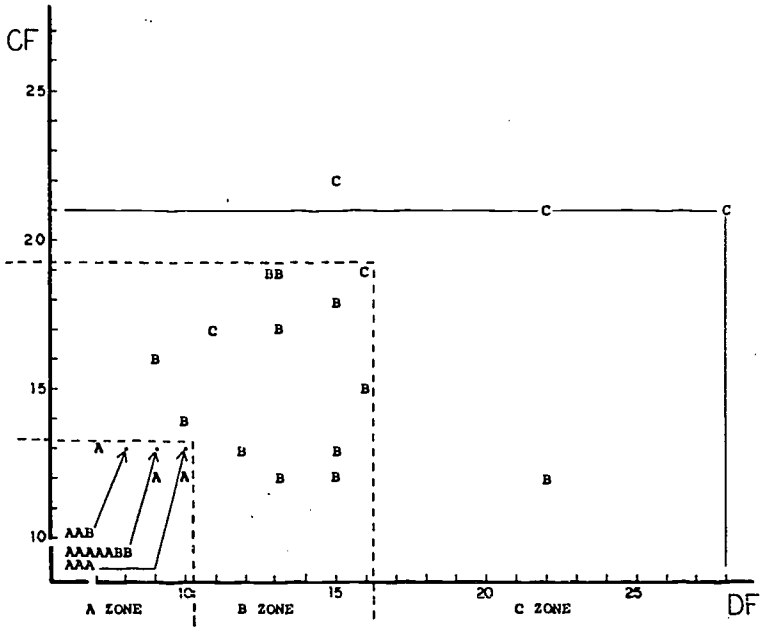

CF
25
20
15
10
AAB
AAAAABB
AAA
A ZONE
B ZONE
C ZONE
DF

FIG. 4

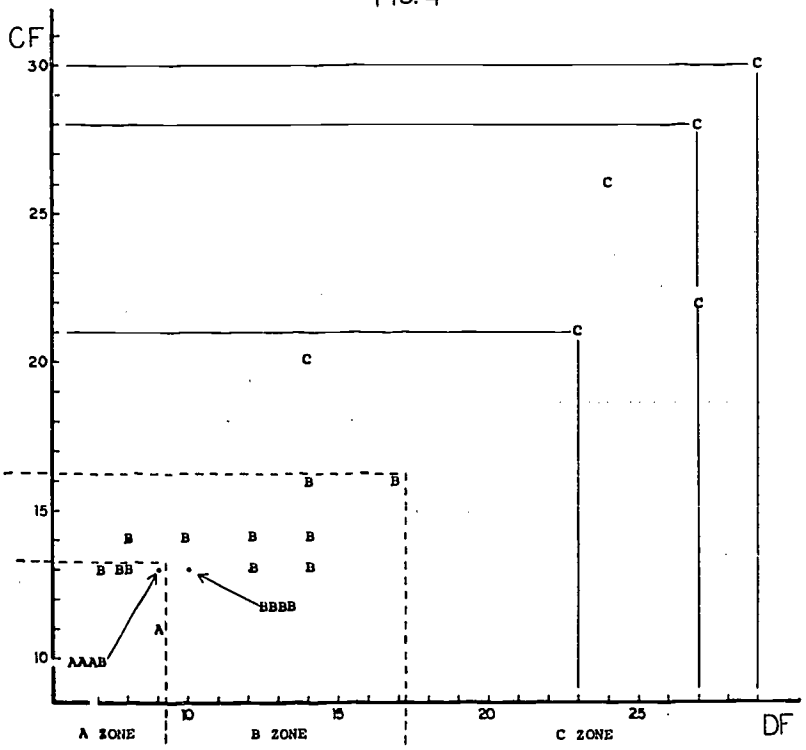

CF
30
25
20
15
10
BBBB
AAAB
A ZONE
B ZONE
C ZONE
DF

TABLE 3

	V (see Fig. 3)	CF (see Fig. 2)
A-Zone	13 A's, 9 B's	13 A's, 8 B's
B-Zone	6 B's, 4 C's	7 B's, 2 C's
C-Zone	1 C	3 C's

	DF (see Fig. 2)	CF & DF (see Fig. 2)
A-Zone	13 A's, 5 B's	13 A's, 3 B's
B-Zone	9 B's, 3 C's	11 B's, 2 C's
C-Zone	2 C's, 1 B	3 C's, 1 B

TABLE 4

	V (see Fig. 5)	CF (see Fig. 4)
A-Zone	4 A's, 14 B's	4 A's, 10 B's
B-Zone	2 B's	6 B's
C-Zone	6 C's	6 C's

	DF (see Fig. 4)	CF & DF (see Fig. 4)
A-Zone	4 A's, 5 B's	4 A's, 4 B's
B-Zone	11 B's, 1 C	12 B's
C-Zone	5 C's	6 C's

FIG.3

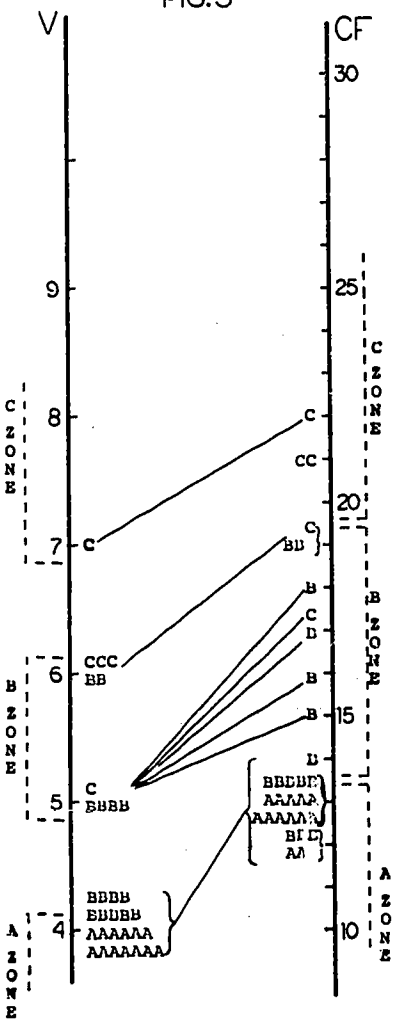
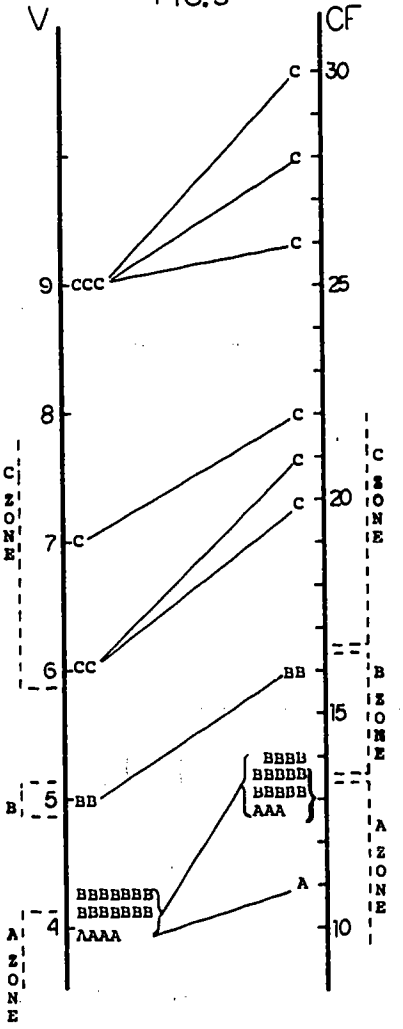


FIG.5



SUMMARY

Programming courses are not only an appropriate environment to perform experiments to test hypotheses and models of program complexity but perhaps the only available environment in which to do meaningful experiments involving the collection of large amounts of data. From such courses we can obtain many programs for the same purpose, in many different computer science subject areas (e.g., numerical analysis, data structures, etc.) and in a variety of programming languages. Many experiments with different models can be performed to rank these programs according to their complexity, and the grade assigned by the instructor would be a readily available independent evaluation of the programs' complexity.

Given the methodological difficulties for studying program complexity that we have mentioned in this paper, we believe we should initially concentrate our efforts on the development and validation of models of program complexity for elementary programs and gradually develop more sophisticated models for larger programs. A large number of elementary programs can be obtained from introductory computer science courses for testing simple models of program complexity which are based, for example, on the complexity of each module of the program. Similarly a fair number of large programs from advanced courses (although smaller than large production systems) can be used for testing models that, for example, not only take into account the complexity of each module but also module interactions, appropriateness of data structures, clarity of design, etc. We believe that accomplishing these steps in the study of program complexity are essential before we attempt to develop absolute measures for the complexity of large production systems.

From a didactic point of view, the use of objective program complexity measures that can complement program correctness models is of interest in itself. The guidelines or feedback provided by good measurements together with good diagnostic messages could provide more meaningful commentary for students on their programs than they are accustomed to getting from instructors or teaching assistants. This point suggests one final--and potentially major--benefit of a software quality measurement system based on the notion of quality relative to an instructor's solution and/or the other students solutions to a programming problem [17]. With the rapid increase in demand on computer science faculty coupled with a decreasing supply of both faculty and graduate students (i.e., teaching assistants), any system with the potential to supply automatic grading not just of a program's correctness but also of its quality could be a major boon to computer science departments everywhere.

REFERENCES

[1] Basili, Victor and Reiter, Robert, Evaluating Automatable Measures of Software Development, Workshop on Quantitative Software Models, IEEE catalog #TH00 67-9, 1979, 107-116.

[2] Brooks, Ruven, Studying Programmer Behavior Experimentally: The problems of Proper Methodology, CACM, Vol 23, 1980, 207-213.

[3] Brooks, Ruven, Using a Behavioral Theory of Program Comprehension in Software Engineering, Proc. 3rd Int. Conf. on Soft. Eng., IEEE, 1978, 196-201.

[4] Chapin, Ned, Data Accessibility in Structured Programming, NCC, 1978, 597-603.

[5] Curtis, Bill, In Search of Software Complexity, Workshop on Quantitative Software Models, IEEE catalog #TH00 67-9, 1979, 95-106.

[6] Dunsmore, H., and Gannon, J., Data Referencing: An Empirical Investigation, Computer, Vol. 12, 1979, 50-59.

[7] Gordon, Ronald, Measuring Improvements in Program Clarity, CSD-TR-268, Purdue University, 1978.

[8] Gould, J. D., and Drongowski, P., An exploratory study of computer program debugging, Human Factors 16, 1974, 258-276.

[9] Green, T., Conditional Program Statements and Their Comprehensibility to Professional Programmers, J. Occup. Psychol., Vol. 50, 1977, 93-109.

[10] Love, T., An Experimental Investigation on the Effect of Program Structure on Program Understanding, Proc. Reliable Soft, Conf., SIGPLAN, March, 1977, 105-113.

[11] McCabe, T., A Complexity Measure, IEEE Trans. on Soft. Eng., Vol. SE 2, 1976, 309-320.

[12] Miller, L., Behavioral Studies of the Programming Process, IBM Research Report RC 7367 (#31711), 1978.

[13] Oviedo, Enrique, Control Flow, Data Flow and Program Complexity, Proc. Computer Software and Applications Conference, IEEE, 1980, 146-152.

[14] Ralston, Anthony, Mathematics and Computer Science, Research Directions in Software Technology, The MIT Press, 1979.

[15] Shneiderman, Ben, Software Psychology, Winthrop, 1980.

[16] Simon, Herbert, The Sciences of the Artificial, MIT Press, 1969. 9, 1979, 107-116.

[17] VanVerth, P.B., and Ralston, A., A System for the Automatic Grading of Programming Style, Proc. NECC, June 1983.

[18] Wirth, N., Algorithms + Data Structures = Programs, Prentice-Hall, 1976.

TEACHING A SOFTWARE ENGINEERING CLASS USING AN IBM PERSONAL COMPUTER (tm)

RONALD I. FRANK
(Visiting Lecturer)

Framingham State College
Computer Science Department
Framingham, MA 01701

We outline the successful use of an inexpensive personal computer to support the programming project portion of a standard Software Engineering Course. First, as background, we specify the environment, i.e. our institution and the equipment we usually use. Second, we outline the course, its purpose, its level, the type of students we have, and their background. We thus provide a reference for the following discussion of the student projects. Third, we discuss the projects themselves as actually accomplished on the IBM PC. Fourth, we discuss the pros and cons of using such a "small" machine as a project development vehicle. Fifth, and last, we review the lessons we have learned from this educational experiment.

A cost estimate sheet is provided showing a cost of computing of about $40 per student per semester for the complete machine service for this Software Engineering course, assuming that the entire cost of the PC was borne by this one course. This cost includes an all-points-addressable color graphics configuration supporting extended Pascal with separate compilation, a full macro assembler, 128KB of main memory, and an 80 cps matrix printer.

I). The Environment (School and Equipment)
Framingham State College (FSC), Computer Science (CS) Department.

FSC is a small (3100 students) four year liberal arts college which is part of the Massachusetts State College System (9 colleges). Massachusetts also supports a University System. At the time of this course (Fall 1982), the FSC CS Department has one full time faculty member (the chairperson) and 27 part-time adjunct faculty members from industry who teach one or two course sections per semester. The department supports 206 CS majors and other students as well.

The home offices of Digital Equipment Corp., Data General, and Prime Computer are within a ten mile radius of FSC. Most of the Route 128 complex is less than twenty miles away. Wang is about 40 miles away. IBM has a strong marketing presence in this area, along with its Cambridge Engineering and Scientific Market Support Center and Cambridge Scientific Center. From these demographic facts derives the unusually strong and experienced faculty we enjoy. To accomodate this faculty, many "day" sections are taught at night, as was this course.

The equipment we use includes a large CDC Twin Cyber 172 System located in Boston, 30 miles away. This had been our traditional support machine for the Software Engineering course. We access it via local terminals running at 300 and 1200 baud. It was reconfigured to 1200/300 from 300 only, and was quite unreliable during the semester this course was given. This encouraged us to use the IBM PC.

We have a DEC PDP-11, a PRIME 400, and a Wang VS-80, locally. The Wang is restricted to administrative use and word processing. We recently purchased an IBM Personal Computer (tm) with 128K (64K on a QUADRAM QUAD (tm) Board) and an 80 cps matrix printer. We use the color/graphics adapter and an Electrohome green phosphor monitor as our standard display (we would now opt for the more expenseive but higher quality IBM monochrome monitor). We have two single-sided 160KB disk drives. Our hardware includes an FSC audio-visual department (shared) color TV for color graphics output and an Electrohome green monochrome projection TV for classroom presentations (projecting the PC display screen onto a regular flat suspended movie screen). The color TV has "video-in" so we drive it without an "rf" modulator thus getting better quality color in "medium" (320 by 200) resolution. All three TV's are driven by the color/graphics adapter.

Our software includes the PC Macro Assembler (1), Pascal (2) BASIC (3), and the BASIC compiler (4), all running under PC-DOS 1.1 (5). Advanced, graphics, BASIC comes with DOS. All programming was done with EDLIN (5) which comes on the $40 DOS disk. DEBUG (5), a single-step debugger and unassembler, also comes on the DOS disk.

II). THE SOFTWARE ENGINEERING COURSE

As part of the four year CS curriculum, we teach a contemporary Software Engineering Course. The class is comprised of third and fourth year students, and is taught by a practitioner with the purpose of introducing them to the Software Life Cycle, the rationale for the various documents required, and the literature of the field. The students get some practical advice from "war stories" and also experience the problems of a team project. They write the documents and perform all the functions required in a software development project including generating requirements, generating time and cost estimates, meeting deadlines, and giving oral presentations.

Structured modular programming is taught throughout the four year curriculum. This course is a review and application of what the students have been taught, but in the context of Structured Design.

Walk-throughs and reviews occur in class and during team meetings. Team meetings occur outside of class. More time is spent in teamwork than in class or in class preparation.

The class used Brooks (6), and Pressman (7), as required texts with Myers (8) as a recommended reference. A number of class handouts were used including a MIL-STD for software development (9), the life cycle document outline from a computer manufacturer, and various reprints from the IEEE "Transactions on Software Engineering". A reserve reading list was maintained at the library. The books included many alternate Software Engineering texts by authors such as Constantine, Myers, Jensen, and Orr. A number of the IEEE tutorials were also on the list.

The course met for two hours, twice a week, for one four-month semester. There were eighteen students, including two professionals and 3-5 others currently working part-time in programming in the Route 128 area. The students had all previously taken one year of programming including Pascal and FORTRAN. They had taken a Data Structures course and had experience using files. Many had had Programming Languages (covering the theory of computer languages) which included a large individual programming project. All had completed an assembler based Machine Architecture Course. Various other courses were shared by the students as a common background, but not a microcomputer course per se. Our CS students have college entrance examination SAT averages of about 1075 (math + verbal).

III). THE SOFTWARE ENGINEERING PROJECTS

The eighteen students were divided into three teams of six each. A team was comprised of a leader, a leader back-up person, a recorder, a chief programmer, and two editors. The leader represented the group in class, the recorder kept the group meeting records, and everyone did word processing and coding. Also, everyone kept their own Programmer's Notebook. All documents were worked on together, but a single person had main responsibility for each document. There were six documents:

1. A Proposal (Software Plan)
2. Requirements Specification
3. Architecture/Functional Specification
4. Preliminary Design
5. Detailed Design and Test Plan (and Error Reporting Form)
6. User's Manual and Documented code

The three groups, after two weeks of investigation, decided on three projects. One, not discussed here, was a traditional large machine Pascal-based project on the CYBER System. It was a general purpose grades maintenance, grades computing, and reporting system, with data file encryption for security. It had a menu driven user interface.

The other two projects, reported here, were similar to each other. They provide to PC Pascal most of the extensive macro-level color graphics capability which is inherent in PC BASIC. In both projects, the screen driver is the BAS.COM file from the PC BASIC Compiler which provides the color graphics macro library to compiled BASIC. The projects bridge PASCAL to the same code - they enable a PC Pascal program to generate color or monochrome graphic images. This is not a capability inherent in PC Pascal.

The BAS.COM file is proprietary to MICROSOFT (tm). That implies that absolutely no documentation is available to find out how it works. It is a library of machine language graphics macros with undocumented functions and calling sequences. It is not our purpose here to document this information - doing so might be a copyright violation.

Our purpose is to document the successful use of the IBM PC in the Software Engineering course. The two PC-based projects, totaling twelve people, shared access to our one PC without long waits. Informal self-scheduling sufficed. Our PC is in a controlled, but public, location. It is available M-F, 8:00 AM - 11:00 PM, and on Saturday 9:00 AM - 6:00 PM. Each team has available a set of diskettes. Each member maintains their own work diskette, but not system diskettes such as DOS, Assembler, or Pascal. The compiler, the assembler, all work diskettes and all documentation are checked out from an operator as if they were a restricted-use library reference book. Diskettes are kept in the Computer Center. This means a student gets his diskette and his team's copy of the system software from an installation operator and returns all of it after the session. The students would have significant difficulty making and keeping copies of the fee software. It can be done, but it wasn't.

Since most of the word processing was done on the PRIME, Wang, and Cyber Systems, there was little use of the PC for report writing. Next semester we will plan an experiment where we use the PC for report writing also. There is more than sufficient time available on the PC to allow for this extended use. We plan to use Easywriter 1.1 (11) because it is not expensive but provides sufficient functions for coding and documenting.

No special classes were held on the PC. The students had to learn PC-DOS, PC Macro Assembler, the use of the PC BASIC Compiler, and the Pascal Compiler. we did hold two tutorial sessions on 8088 architecture, the assembler, and the overall structure of the PC Pascal. At times we had to advise some team members on 8088 register use.

Importantly, the PC Pascal is powerful. It is more powerful than the mainframe CYBER Pascal we use. As such, it is a significant item to learn.

The PC Macro Assembler is also a mainframe level system. The teams divided up the labor of learning into individual efforts and then held cross-tutorials for each other. Certain advanced individuals were clearly individually responsible for the major breakthroughs in understanding and problem resolution. Insight in the face of zero documentation clearly varies among individuals by at least a decimal order of magnitude, as measured by the time needed to solve a problem.

IV). PROS AND CONS OF THE PC USE

There are no significant differences between the capabilities of the IBM PC with its languages, and large systems with their languages, except that the PC has wider capabilities, such as color graphics, at a much lower per student cost of computation.

Cost is the main factor. We supported twelve students doing a major software effort with no strain on the PC schedule. All this for less than $4700, as a one-time-charge. (See the cost sheet at the end of this article.)

The Pascal reference manual needs a tutorial manual. The assembler also needs a tutorial but to a lesser degree than Pascal. We can't expect BAS.COM documentation, but it would be helpful. The portable Electrochrome TV unit, driven off of the graphics adapter, enabled us to have a large screen for in-class presentations. The portability of the PC enabled in-class presentations. The fact that the PC color graphics could use an ordinary A-V color TV set meant that we could afford to do a color graphics project.

The completely open (i.e. fully documented) nature of the IBM PC system means that we can assign such "real-world" system design problems and get them done within the confines of a single semester. the Technical Reference Manual (10) for the PC explains the hardware in complete detail. The fact that native BASICA and the BASIC Compiler are essentially identical means that we could run color graphics tests interpretively, and then compile to find links to BAS.COM in order to isolate its functional routines.

Development would have been faster if we had bought the double-sided (320KB) disk drives. That would have reduced the "Floppy Flipping" necessary to do a Pascal compilation or an assembly. It would be ideal if our students had these machines at home. That would make it easier and more productive for them. One student did have an IBM PC at home. However, by chance, he was on the CYBER project!

The reliability of the hardware was perfect. The software was also reliable, but we suspect certain difficulties came from either poor documentation in the Pascal manual or from true bugs. We couldn't spend the time to isolate the problems. The students found fixes or bypasses that worked. None of the problems encountered left "time bombs" in the codes. The codes came out "clean".

V). LESSONS LEARNED IN THIS LIMITED EXPERIMENT

Even the smallest schools can now provide their CS students with all of the hardware and software tools for real-world projects. This new 16 bit micro, for example, provides large systems capacity, capabilities, problems, and opportunities at a very low per-student cost. Very inexpensive, technically advanced bit mapped graphics is a major innovation.

Physical and media security is not a major problem. Student enthusiasm is a major plus.

The Programmers' Notebooks that the students maintained became so valuable to their work that the students didn't want to hand in the notebooks for instructor review until the coding and documenting were completed!

VI). ACKNOWLEDGEMENT

I want to thank Anita Goldner, FSC CS Department Chairperson, for supporting this experiment and participating in it. Also, I want to thank Paul Ferguson, FSC Computing Center Director, for handling all the administrative details needed to order, set up, and run the open-shop PC. My thanks to the class for their help also.

COST SHEET

For Computer Components Only

$4700 Purchase (Including Software) minus cost of capital and maintenance which comes to:

$1175/year for 4 years
$118/month for a ten month year
$.33/hour for 84 hours a week availability

Twelve students, not doing word processing on the PC itself, had little conflict using one PC. With word processing there would be a need to schedule usage, but the system (1 PC) would still not be saturated. We estimate that fifteen students, each needing five hours per week on the system (75 hours out of 84 available) would saturate the system. If the Center hours were extended to include late night and Sunday use, we estimate that twenty students could be supported on one machine for this course.

At fifteen students per machine, the total cost is $7.80/student/month. This includes all system components, but not room rent or security supervision. By opening the center for more hours, enabling 20 students to use the PC, the cost drops to about $5.90/student/month. This comes to $40 per student or $30 per student, respectively, for a semester (1/2 year). This estimate does not include media, supplies, or a formal maintenance contract. Media and supplies come to much less than $50/team/semester. The 4 year life time assumed for the PC is reasonable due to technological progress calling for its replacement. The machine will last and be useful much longer than that. A maintenance contract costs approximately 10% of system cost/year.

BIBLIOGRAPHY

(1) IBM Personal Computer Macro Assembler. #6024002.

(2) IBM Personal Computer Pascal Compiler. #6024010.

(3) IBM Personal Computer BASIC. #6025010.

(4) IBM Personal Computer BASIC Compiler. #36024003.

(5) IBM Personal Computer DOS. #6024001.

(6) Brooks, F., The Mythical Man Month, Addison-Wesley, 1975.

(7) Pressman, R. S., Software Engineering, McGraw-Hill, 1982.

(8) Myers, G., Software Reliability, John Wiley, 1976.

(9) MIL-STD 1679 (Navy), Military Standard Weapon System Development, Chief of Naval Material, NAVMAT 09Y Wash., DC 20360, 1 December 1978.

(10) IBM Personal Computer Technical Reference Manual. #6025005.

(11) IBM Personal Computer Easywriter (1.1). #6024005.

CRISIS IN PROGRAMMING, OR HISTORY DOES REPEAT ITSELF

Jacques LaFrance

Department of Mathematical Science
Oral Roberts University, Tulsa, Oklahoma.

Abstract

The increasing cost, unreliability, complexity and unmaintainability of programming efforts of the 1960s gave rise to the discipline of structured programming. We face a similiar crisis in microcomputer software today because of a new generation unfamiliar with the past. New approaches to introducing programming are needed to solve the problem. A programming language called Antfarm has been developed to introduce structured programming logic in such a simple and entertaining format that even young children can begin to learn the currently best accepted principles of program design. The program has been used with kids from age 5 to college students and adults who are either new to programming or have only worked with BASIC.

Historical Background

The Software Crisis of the 1960s.

As computer systems grew from the early days of computing, a crisis was reached sometime in the mid 1960s when the cost of software became noticably larger than the cost of hardware and the logical complexity of software systems became unmanagable. In fact some systems became so complex and interconnected that someone once calculated that any changes introduced to correct errors had such a large probability of introducing other errors that it was very unlikely that the system would ever be correct.

Leaders in the industry began desparately seeking a solution to this crisis, a solution that would give the field of software design a precision similar to what was being achieved in the hardware area. What was needed was a methodology for software development which would allow it to be accomplished efficiently and would enable the logic of complex systems to remain manageable.

A Solution to the Software Crisis.

A paper by Bohm and Jacopini in the Communications of the A.C.M. in 1966[1] provided a foundation for a new discipline of programming. They proved that any logical system can be reduced to a combination of three basic logic forms. These forms are called sequence, selection, and iteration. A sequence is one operation after another; a selection is the making of a choice between two or more alteratives; and an iteration is the repetition of something until some terminating condition is met.

More complex logic is created by including these structures within each other. For example, consider an algorithm in which the main logic structure is a sequence of three things, the first of which is a selection and the last an iteration. The selection chooses between two sequences, each of which contains an iteration, which in turn contains a sequence, one element of which is a selection, etc. At each level, the structure contains only these three basic forms. Each of these contains one entry and one exit, the logic cannot wander off in some hard to follow path. The discipline of limiting one's logic design to these simple structures is called "structured programming," and the programs which result are called "structured programs." Other concepts that have developed in conjunction with structured programming are "top-down design", "structured design", "Chief Programmer Team", "structured walk-throughs", and "proving program correctness," some of which are often included in the meaning of structured programming. For a more complete discussion of these and examples of how they have improved software development, see McGowan and Kelly, Top-Down Structured Programming.[9]

These new programming disciplines have become generally widely accepted in industry and higher education. The Pascal programming language was developed by Niklaus Wirth around 1970 to give a tool for better teaching of modern programming concepts than was possible with existing programming languages. In particular, it was designed to be able to teach structured programming concepts in a natural way.

History Repeats Itself

The New Software Crisis.

Now we are entering the software crisis of the 1960s afresh. The problem we now face is that the widespread multiplication of personal computers in the last 5 years, all of which come equipped with some version of BASIC as the default (or in some cases, the only) programming language, is leading to a new generation of software developers unfamiliar with the above history. BASIC was developed in the early 1960s prior to the advent of modern structured programming principles, and

therefore these principles had no influence on its design. Although one can do structured programming with any language, BASIC does not naturally encourage structured programming and naturally leads the novice programmer away from structured programming principles. Its design structure encourages obtuse linear low-level coding with arbitrary branching.

Since the distribution of personal computers has exceeded the computer professional community's ability to provide adequate training, many of the new users are being self-taught or are being taught by others who are largely self-taught. This is leading to a generation of new programmers who are not aware of the progress the field has made in software design principles in the last twenty years. As George Santayana said once, "Those who cannot remember the past are condemned to repeat it."[15]

A Solution to the New Crisis.

One solution to this crisis is to develop tools for teaching structured programming concepts on microcomputers to all age levels, especially the younger children who will soon become the next generation of programmers. The concepts of structured programming or structured programs are not difficult and could easily be understood by children if given the right tools. The basic logic structures of sequence, selection, and iteration are understood by children before they reach elementary school. Consider the following sentences: "Put your toys away, put on your pajamas, and then we will read a story," a sequence; "If the weather is nice on Saturday, we will have a picnic," a selection; and, "Write your spelling words 10 times each," an iteration.

Over the past three years we have been working on such tools.[6] A special language for use in teaching structured programming concepts to children has been developed. This language, called Antfarm, has been designed according to the following criteria, which, if met, could make it a very useful tool to introduce structured programming concepts:

1. The language would only include the basic structures of sequence, selection, and iteration, i.e. no unconditional transfer ("GOTO"), but would include a complete set of these basic structures.
2. The language would be as close as possible to the English with which the children were already familiar, and the vocabulary would consist of short simple words learned early in school.
3. The language would focus on program logic structures, which are deemed harder to understand, by eliminating any consideration of data structures.
4. The situation of the language would be fun and imaginative, providing high motivation to work with the it.
5. The language processor would be implemented in as transportable a way as possible.

The Antfarm language uses the text screen and therefore is not limited to hardware with special graphics. The processor has been written in UCSD Pascal and therefore will run on the widest variety of processors and machines. A Forth version is also being developed for use on smaller machines. The screen is a farm on which an "ant" lives and raises food. Everything the ant does uses energy and therefore it must eat some of its crops to stay alive. It can plant interestingly shaped fields, dance, march, explore, find its way around, and so has proven to be a highly interesting environment in which the children can work. Whereever we have used it, the children have been very intrigued and highly motivated. Goals 4 and 5 haved been achieved successfully. The following section shows how the first three goals were achieved.

The Antfarm Language

The Antfarm Setting.

With Antfarm, the screen is a farm on which an ant colony lives and raises food. The ants consume energy and must be fed to stay alive. The ants and the other symbols are all made of typewriter characters and do not restrict the language to use on special graphic hardware. All movement is in terms of the rows and columns of the text screen. The ants can dance, march, explore, have lunch, and draw rudimentary pictures by planting seeds. One student even programmed a hurdle race with three ants, and in another program an ant finds its way through a maze.

The ant looks like this, depending on orientation, where the "*" is its head:

```
\*/     *1      \ /
 X      -X     *XO-
 0       0-    / \
/1\      1\
```

Figure 1

The body characters represent the ant's weight, energy supply, and change with each command. "XO" is the maximum, and it represents 400 units of energy, or enough to do 400 commands. The seeds grow into plants as follows:

```
.  seed
,  germinated seed
;  starting to come up
!  young sprout
Y  tall plant with branches
P  flower
@  full grown mature food
```

Figure 2

The Antfarm Commands.

Basic Commands. The basic commands in Antfarm are MOVE, one row or column forward, BACKUP, one row or column backward, TURN LEFT, rotate 45 degrees to the left, TURN RIGHT, rotate 45 degrees to the right, PLANT, put a seed down where the head is, EAT, consume whatever is under the head, and WAIT or NOTHING, which has the ant do nothing but use up

one energy unit and spend one time unit.

These commands or any others can be listed one after another making a sequence, such as MOVE PLANT MOVE TURN LEFT.

Program Logic Control: Iteration. Two forms of repetition are allowed, the common counting loop and the conditional loop. Both begin with the word DO or the word REPEAT and end with the condition for loop termination. In the case of the counting loop, this is a natural number followed by the word TIMES. In the case of the conditional loop, this is either the word TO or the word UNTIL followed by the condition. The ant has a rich set of conditions to chose from, which contributes to the captivation of the language and its power in illustrating program logic structures. These conditions include the ability to check position, e.g. ROW number, PAST COLUMN number, NOT PAST ROW number, orientation, e.g. FACING N, NOT FACING SE, appetite, e.g. STUFFED, NOT HUNGRY, smell, e.g. SMELL FOOD, NOT SMELL DIRT, and sight, e.g. SEE PLANT LEFT, NOT SEE JUNK AHEAD. In addition conditions may be combined with AND or OR.

Program Logic Control: Selection

Selection is implemented with the same word most common in English and other programming languages, IF. There are two forms, "IF condition THEN commands END-IF", and "IF condition THEN commands IF-NOT commands END-IF". In the latter, ELSE, IFNOT, and OTHERWISE are accepted as synonyms for IF-NOT. In either form a comma is a synonym for THEN and a semicolon or REGARDLESS is a synonym for END-IF. The condition is exactly the same as the condition allowed for loop termination. Two examples of selection commands are:

```
IF FACING N , TURN LEFT ELSE TURN RIGHT ; MOVE MOVE
IF HUNGRY AND SMELL FOOD THEN EAT REGARDLESS MOVE
```

Program Logic Control: Loop Exit. Another flow of control structure is STOP, which terminates the loop if it is used in one. An example of use is:

```
DO MOVE IF NOT SMELL FOOD , STOP ; EAT 10 TIMES
```

which is a counting loop with the possibility of premature exit due to running out of food to eat before the count is exhausted.

Subprogram Definition. Another major consideration in modern program development is that of building the total program out of relatively small modules. The main program is simply a set of logic structures to put the major modules together in their logical order. Each of these modules is also just the set of logic structures needed to combine the modules out of which it is made, and so forth, until the structures being combined are simply fundamental commands in the language.

Antfarm forces the development of modular programs by restricting each module to no more than 80 characters, one line on a crt or two lines on an Apple II. The facility provided for building program modules is that of teaching the ant new commands. The format of this command is "LEARN command-name definition". An example is:

```
LEARN REVERSE DO TURN RIGHT 4 TIMES
```

The command the ant is to learn is named REVERSE. After the ant is given the command REVERSE, it will perform DO TURN RIGHT 4 TIMES.

Top-down Design in Antfarm. Since words do not have to be defined before they are used in a definition, the ant can be taught a complex program by the methodology of top-down design. The following is an example of teaching the ant to plant a garden, which has the shape of the outline of a square, by the principle of top-down design:

```
LEARN SQUARE DO SIDE 4 TIMES
LEARN SIDE ROW-OF-PLANTS CORNER
LEARN ROW-OF-PLANTS DO PLANT MOVE 5 TIMES
LEARN CORNER TURN RIGHT TURN RIGHT
```

After teaching the ant all of these commands, the ant may be instructed to to make a square simply by typing the command SQUARE.

Antfarm Examples.

The next program is an example in which the ant makes itself oriented toward the north, pointed toward the top of the screen:

```
LEARN FACE-NORTH IF NOT FACING N THEN
   TURN-NORTH ;
LEARN TURN-NORTH IF FACING-EASTERLY THEN
   TURN-TO-LEFT IF-NOT TURN-TO-RIGHT ;
LEARN TURN-TO-LEFT DO TURN LEFT UNTIL
   FACING N
LEARN TURN-TO-RIGHT DO TURN RIGHT
   UNTIL FACING N
LEARN FACING-EASTERLY FACING NE OR
   FACING E OR FACING SE
```

In this example, both the "IF condition THEN command" and the "If condition THEN command IF-NOT command" forms were used. Also this command illustrates that named conditions can be learned, FACING-EASTERLY in the example. This allows a powerful hierarchy of logical conditions for program structure control and provides a basis for introducing the concept of a function in the student's subsequent education.

The following is another example in which the ant finds its way through a maze using the algorithm of always staying against the left wall. The maze is made of plants just two columns apart so the ant can use its rather near-sighted vision to see the walls. There is food at the end to signal completion of the maze.

```
LEARN FOLLOW-LEFT-WALL DO GOLEFT
   UNTIL OUT-OF-MAZE
LEARN GOLEFT LOOK-FOR-LEFT-GAP
   MOVE MOVE AGAINST-WALL?
LEARN LOOK-FOR-LEFT-GAP IF NOT
   WALL-ON-LEFT THEN TURN-TO-LEFT ;
LEARN TURN-TO-LEFT DO MOVE MOVE TURN
   LEFT TURN LEFT UNTIL WALL-ON-LEFT
LEARN AGAINST-WALL? IF WALL , DO TURN
   RIGHT TURN RIGHT UNTIL NOT WALL ;
```

This example shows powerful uses of selection and iteration in expressing program logic. Notice the form "IF X THEN DO SOMETHING UNTIL NOT X" in the definition of AGAINST-WALL? This is equivalent to the form, "WHILE NOT X DO SOMETHING", which illustrates the ability to build different logic forms from a few simple ones. The former is probably more common in natural language, as in, "When the bell rings, if you haven't finished, keep working until you are finished," and, "After school, if it is raining, wait there until it stops raining." This example also shows the benefit of mnemonic names being used, especially the names used for the learned conditions, OUT-OF-MAZE, WALL-ON-LEFT, and WALL. These are defined as follows:

```
LEARN OUT-OF-MAZE SEE FOOD AHEAD
LEARN WALL-ON-LEFT SEE PLANT LEFT
LEARN WALL SMELL PLANT
```

This illustrates that not only can the ant be taught command sequences as modules, but it can also be taught conditions as named modules as well. This in a sense gives both procedures and functions, although functions are limited to type Boolean.

Antfarm for teaching structured programming.

These few examples illustrate the potential of Antfarm to teach the basic concepts of structured programming. As we have used Antfarm, we have found it to be very effective in teaching concepts of structured programming, not only to children but also to college students and adults. Other professionals familiar with structured programming have also been impressed with its potential when they have had the opportunity to observe Antfarm.

Experience with Antfarm

An Elementary School Antfarm Class.

In the fall of 1981, Antfarm was used one day a week for eight weeks at Grissom Elementary School in Tulsa, Oklahoma. There were eight classes, each limited to 10 students on a first come first served basis as part of an enrichment program available to all students. The classes were about 35 minutes long. One computer was available and that was one the author brought to the school for the day and took back home after the last class. Because of the severely limited computer time available, the instructor did all the typing and went over the students' programs verbally when there were serious shortcomings the students needed to correct.

The students all responded very enthusiastically to this project and began to grasp the concepts of program structure, even though the computer time was severely limited. The best work was done by a student who created a program of about 19 modules which made the ant draw (plant) the shape of a Christmas tree and then wait for it to grow up. He only needed help on the latter part and two simple errors. It clearly showed top-down structure. Also one student compared notes with his older brother who was taking a high school BASIC class. They agreed that the younger one had learned more about programming with Antfarm than the older one had with BASIC.

Use of Antfarm in Summer Camps.

A Two-week Camp at ORU. During the summer of 1982, the Oral Roberts University School of Education ran two two-week camps for kids from 5 to 17 years of age using their new microcomputer laboratory. Each student had 15 hours in the camp, half of which was spent on the computer and the other half receiving instruction. All students began with Antfarm; however, for the second week, the older students switched to a Pascal system called Computer Power, produced at the University of Tennessee under the direction of Prof. Mike Moshell in order to provide materials for teaching Pascal in high schools.[11]

The two five-year-olds that attended learned something about the preciseness of computers. They were able to give the ant a sequence of commands to make it do what they wanted and quickly saw the consequence of wrong commands when the ant did something different from what they had planned. However, they were not able to get to the point of being able to compose learned programs. This level of abstraction requires a more mature mind. We found that at about eight years of age the children seem to have developed enough of an idea of abstraction to be able to define simple programs for the ant.

All students, no matter how old, definitely learned some good foundational principles with Antfarm. The older ones seemed to be able to carry the ideas from Antfarm over to their work with Pascal quite easily, although, we believe more time spent with Antfarm would have benefitted them. The results of this experience confirmed the previous results: they enjoyed it very much and were able to put together logical program sequences of varying complexity.

A Summer Camp Conducted at OU. The University of Oklahoma College of Education also conducted a computer camp in the summer of 1982, under the direction of Evelyn Gatewood. The students in this camp attended 3 hours each day for two weeks. During this time they were given experience with BASIC, Antfarm, and LOGO. In the Antfarm segment, the students, especially the younger ones, seemed to grasp the concepts more quickly than with either of the other two languages. The vocabulary and language structure was simpler and more easily understood. It was noted also that the children were more fascinated by having an "animal" to take

care of than by just making pictures. Some of the children became very protective of their ant, making sure it didn't starve, sometimes by overfeeding it, as perhaps they are apt to do with household pets as well. Taking a personal interest in their ant made the children more motivated and seemed to provide better communication between the child and the computer.

Results with College Students.

We have also used Antfarm for a period of about one week in our Introduction to Computing class at Oral Roberts University. We have found that this experience helps the new students understand the concepts of program structure better prior to studying it in Pascal. It also seems to serve as a good tool to wean students who have programmed in BASIC prior to entering the course away from the unmodular, low-level, arbitrary branching style of programming to which they had become accustomed. In both cases the students were more quickly able to begin to develop modular structured approaches to program design than without the brief experience with Antfarm.

Other Languages for Teaching Structured Programming.

There are several other languages that can be used to teach structured programming, including Pascal, Karel the Robot, and LOGO. Pascal was designed by Niklaus Wirth[5] specifically to teach programming. It remains the primary tool for this in higher education. Computer Power, developed by Mike Moshell, Robert Aiken,[10,11] and a team of others at the University of Tennessee, makes Pascal suitable for teaching at the high school and possibly junior high school level.

Karel the Robot was developed by Richard Pattis[14] to introduce structured programming to his university classes in very much the same way as we have used Antfarm. Like Antfarm, Karel the Robot leaves out the concept of data structures in order to focus attention on the logic structures, and the language consists of commands to a robot to move around it's environment. It implements only structured programming constructs, and gives a graphic situation in which the student can see the effect of his commands on the the Robot.

LOGO was developed by Seymour Papert[12,13] as a tool to use with children to study their patterns of problem-solving. Although it was not designed for teaching programming, it does contain much of the structures needed to teach structured programming.

In comparing Antfarm with these, the following points can be made. Antfarm is simpler than any of these, yet gives more powerful program logic structures. Also, it is more English-like and gives the student a more immaginative environment in which to work. The simplicity and immaginativeness makes it more suitable with younger children than any of the others, and the powerfulness of it's logic structures make it a challenge to older minds. The fact that it forces the learner to make modular, structured programs is the key to enable it to contribute to the solution of the new software crisis.

Conclusions

It is possible to teach structured programming to children in a way that they enjoy and which lays a good foundation for future programming. We have seen Antfarm do this. Antfarm does indeed achieve the goals set out for its design. It is fun; it allows only structured programs; it is based on simple English words; and it is simple enough for young children. It is our hope that Antfarm, or things like it, can achieve widespread use as introductions to programming and forestall the present crisis in programming by making it possible for potential future programmers to begin their computer careers learning to make modular structured programs.

References

1. Bohm, C., and Jacopini, G., "Flow diagrams, Turing machines and languages with only two formation rules," Communications of the ACM, vol. 9, no. 5 (May 1966), pp. 366-371.

2. Cohen, Harvey A., "Oznaki and Beyond," Proceedings of the National Educational Computing Conference, Iowa City, IA, 1979, pp 23-24.

3. Dahl, O. J., E. W. Dijkstra, and C. A. R. Hoare, Structured Programming, Academic Press, New York, 1972.

4. Dijkstra, Edsgar, "GoTo Statement Considered Harmful," Communications of the ACM, Volume 11, Number 3 (March 1968), pp. 52-54.

5. Jensen, Kathleen and Niklaus Wirth, Pascal User Manual and Report, Springer-Verlag, New York, 1973, 1976.

6. LaFrance, Jacques, "Shall We Teach Structured Programming to Children?", Proceedings of the National Educational Computing Conference 1980, Norfolk, VA, 1980, pp. 261-265.

7. LaFrance, Jacques, "Reorienting Students to Structured Programming with Antfarm", Proceedings of the 1982 Western Educational Computing Conference, San Diego, CA, 1982, pp. 35-42.

8. Lieberman, Henry, "The TV Turtle: A LOGO Graphics System for Raster Displays," MIT, A. I. Memo 361, June 1976.

9. McGowan, Clement L., and John R. Kelly, Top-Down Structured Programming Techniques, Petrocelli/Charter, New York, 1975.

10. Moshell, J. M., G. W. Amann, and W. E. Baird, "Structured Gaming: Play and Work in High School Computer Science," Proceedings of the National Educational Computing Conference 1980, Norfolk, VA, 1980, pp. 266-270.

11. Moshell, Michael, Proj. Dir., Computer Power: A First Course in Using the Computer, Gregg Division, McGraw Hill, New York, 1982.

12. Papert, Seymour, "Teaching Children Thinking," MIT, A. I. Memo 247, October 1971.

13. Papert, Seymour, Mindstorms: Children, Computers, and Powerful Ideas, Basic Books, Inc., New York, 1980.

14. Pattis, Richard E., Karel the Robot: A Gentle Introduction to the Art of Programming, John Wiley & Sons, New York, 1981.

15. Santayana, George, The Life of Reason, pg 284.

16. Solomon, Cynthia J. and Seymour Papert, "A Case Study of a Young Child Doing Turtle Graphics in LOGO," MIT, A. I. Memo 375, July 1976.

17. Tomek, Ivan, "Josef, Programming for Everybody," ACM SIGCSE Bulletin, Volume 14, Number 1 (February 1982), The Papers of the Thirteenth SIGCSE Technical Symposium on Computer Science Education, Indianapolis, IN, February, 1982, pp. 188-192.

18. Watt, Daniel H., "A Comparison of the Problem-Solving Styles of Two Students Learning LOGO: A Computer Language for Children," Proceedings of the National Educational Computing Conference, Iowa City, IA, 1979, pp. 255-260.

AN EVALUATION OF A LOGO TRAINING PROGRAM

M. Elizabeth Badger

Massachusetts Department of Education

This paper describes the evaluation of a computer-based school program which utilized the turtle geometry and sprite functions of the Logo language. The study used both observational and quantitative methods to measure the effect of the program on: 1) pupils' familiarity with basic geometric concepts; 2) pupils' understanding of Logo commands; and 3) the relationship between the two. Attitudinal and organizational factors were also considered. Although generally positive results were noted, the author expresses some reservation about the cognitive benefits of unstructured activities. She suggests that the strong visual appeal of the Logo programs may obscure their potential use as "problem-solving environments".

In Mindstorms, Seymour Papert[1] argues for a new conception of education and mental growth. Given the opportunity to use computers and, more particularly, to use the MIT-developed system called logo, children could develop an intuitive sense of geometry, physics and the procedural thinking that underlies problem solving. Not only would the graphical nature of the system appeal to children's imagination, but it would serve as visual reinforcement and feedback to their mental explorations.

Among educators, there is general agreement that Logo has features that make it especially appropriate as a conceptual tool. In comparison with other readily available languages, it is interactive and well-structured, allowing the user to define and manipulate sub-procedures. It contains good graphics, with very simple body-centered and Cartesian coordinates; and it is designed to be conceptually easy.[2] On the other hand, there is little specific evidence for its effectiveness in the class. The evidence that has been offered has been the product of a special environment, i.e., programs sponsored by the Artificial Intelligence laboratories of MIT and Edinburgh. There exists little guidance for the teacher or administrator who asks the question: What intellectual benefits can I expect to derive from introducing Logo into the ordinary class?

This paper represents an attempt to answer this question. It describes a program that was carried out in two Cambridge, Mass. schools and was conducted by a group of individuals who were not experts in the Logo language. The evaluation itself is primarily illuminative, with some reliance upon test results to confirm its more subjective findings.

The Program

The program took place in the winter of 1982, when the Cambridge School Volunteers proposed to teach a 5-week course in Logo to the 6th grade classes of two Cambridge schools. The majority of the volunteers were Harvard students who were experienced in programming languages; however, few had had previous experience with Logo. In preparation for teaching, they were given a short training course in Logo, as well as various Logo materials that had been produced by the MIT-sponsored project in the Brookline Public Schools. They were also asked to keep a journal of class progress. This was to be used, not only as a vehicle for suggestions and cautions, but to insure a continuity during the program.

The Schools

School A (Classes A1 and A2).

The two schools differed in population, curriculum and environment. In large part, the pupils in School A were newly arrived in the United States and Cambridge. All but 7 of the 32 pupils in the 6th grade were receiving some kind of extra help due to language or learning problems. Because of this, teachers tended to stress the basics. Most of the mathematical teaching concentrated on computation, with little reference to more general mathematical "concepts". The children stayed in their own self-contained classrooms and were taught most subjects by their classroom teacher. As a result, except for the occasional deliverer of a message, there was little movement in the corridors. At School A, the 8 Apple microcomputers were arranged in a rectangular array in one corner of a large library. Each class was brought to the library by its teacher and, for the most part, each teacher stayed to help supervise the work done.

School B (Class B1)

School B is surrounded by Harvard and Lesley College. Part of its student body came from the immediate neighborhood and tended to be the children of professionals; part was brought by bus from other areas of the city. Each teacher among the older children is considered a subject specialist, has covered a wide range of topics with his pupils and was eager to house the 6 Texas instruments microcomputers at the back of the classroom. The 24 6th grade pupils in the class were divided into two groups, and the afternoon of each day was devoted to Logo. During this time, their teacher met his assigned mathematics classes in another room in the building. The 6th grade group that was "off the computer" would either go elsewhere for instruction or would remain in the room with pre-assigned seat work.

The Microcomputers

The two types of microcomputers also differed in ways which had consequences for the teaching in the two schools. The Logo language is built into the TI microcomputers. As a results, pupils at School B had access, not only to turtle geometry which allowed them to draw figures and patterns on the screen, but to the sprite program which permits the programming of motion and speed. On the other hand, they could not easily save their programs. The tape device proved to be a cumbersome procedure and was barely used. Consequently, the procedures for individual programs were usually typed anew at the beginning of each session.

In contrast, Logo is not a feature of the Apples, but must be read in through the use of a disk. This limited the scope of activities for the children in School A because the disk contained only turtle geometry. However, the use of the disk drive had certain advantages. It allowed children to save their own programs on personal disks and allowed for later printout. This feature was used extensively by all the pupils at School A.

The Pre-Test

In order to assess the pupil's familiarity with various mathematical concepts, a test was administered at the start of the program. It was composed of a series of questions concerning area, angles, coordinates and permutations. Standardized mathematics achievement test results were also collected.

From the pretest responses, it was evident that the pupils in the two schools differed greatly in their mathematical knowledge. For example, almost none of the pupils in School A knew what an angle was. When asked to estimate size, many based their judgements on the length of the arms. Most left these questions blank or wrote "don't understand". In contrast, most (91%) from School B correctly drew 90° and 45° angles, and over half were able to recognize a 90° angle that was drawn at an unfamiliar tilt.

In addition to these differences in mathematical experience, there was a definite progression in their standardized mathematical achievement scores. When grade equivalent scores were ranked (a total of 45 were available), the pupils from School B had, on the average, higher scores than those from School A. Within School A, the two classes also differed. The average of the ranks of the three classes are listed below (1 indicates the highest score achieved; 45 the lowest):

Class B	16
Class A1	31
Class A2	23

How the Program Went

The program at School A took place every morning from 11 to 12:30. Each class was allocated 45 minutes, with 2 pupils per computer. There were usually at least 2 tutors present, as well as the class teacher.

Using the model of computer as "tutee", the pupils were encouraged to explore the computer capabilities. They seemed to learn the basic commands very quickly and on the first day the tutors' journal noted, "High optimism. A few know Back. All know Left, Right, Forward." Within a few days, however, some of the tutors seemed perplexed at the amount of raw energy that the computers seem to have released. The pupils themselves were delighted in their new-found power when they found that the computer would respond immediately, and in marvelously unexpected ways. Type in some numbers and commands, and the turtle would zip around the screen, building up a constantly changing pattern that could be changed even more by varying the color.

The tutors on the other hand, worried about the extent of their understanding. It was the holistic impression that seemed to capture the pupils' attention, not the component parts. Sensing their lack of focus, one of the classroom teachers mimeographed sheets listing the basic primitives and some nested repeats. The pupils copied these into their notebooks and on to the computer, but tutors continued to wonder if the REPEAT instruction was understood. The children loved large numbers and used them for everything---REPEAT 765, RT 675, FD 786,etc. It was unclear whether or not they had any conception of the role of each command in this grand design. Their excitement and energy set up so much static that it was hard to see that they were actually progressing intellectually.

In an attempt to focus their attention on the procedures involved, cards giving instructions for making designs were introduced in the third week. These seemed to work fairly well. The children liked them and enjoyed constructing the small designs. Perhaps as a result of the design cards, the tutors noted that about 50% started using procedures and almost all had settled down. Also, during this period each pupil was given a disk on which to record individual programs. These provided a focus for

activity because designs could now be saved. They enjoyed this and began to share programs with each other. Many of their programs were copies of or modeled after the design cards, but some children used the design cards as a take-off point for their own explorations.

However, on the Monday of the fifth and last week, the tutors noted that some were getting bored. "Several didn't even want their disks today. Others seem to need new ideas, to be encouraged to explore further." Someone in the second group found out how to make the computer "beep". This spread and led to intermittent beepings throughout the sessions. The tutors' comments grew plaintive: "Lots seemed bored. We need something new. Perhaps more guided instruction at this point? A lot of use of background color change. Some kids still don't have a grasp on the basic turtle commands. Telling left from right is a problem for at least one of the girls."

School B

The program at School B is less easy to describe. Possibly because the program itself was less structured, it was harder to see where it was going. Membership in the two groups working on the computer was not fixed but changed according to outside scheduling. This caused confusion at the beginning of the sessions, and the tutors noted that the classes tended to be "unruly". At any time during the computer period there were pupils coming in and out, some working at their desks on other projects, some with no discernible work to do. In fact, at times some children did not work on the computer at all during the scheduled period. However, this did not visibly bother them, possibly because there was the opportunity to use the computer at leisure and in relative privacy after the rest of the school had left for the day. In fact, a kind of computer club grew up for projects after school and some of the boys used their knowledge to teach Logo to the 7th and 8th grades. Possibly because of this, set time was less important to the children.

The Post-Test Results

As the program developed, an insistent question began to emerge. That was: how much did the children understand what they were doing? At School A, motivated by the promise of printed results, the children produced a multitude of designs.

At School B, without a means of recording their work, they seemed to have been captivated by the activity of sprites. At the end of 5 weeks of daily work on the computer, did the pupils have a sense of computer programming in turtle language, or had they become mesmerized by the visual potential of the computer?

These considerations led to the construction of a post-test which was composed of four parts. In it pupils were asked to: 1) write a series of turtle commands to produce a given series of figures; 2) draw the figures that would be produced by a given procedure; 3) draw and estimate the size of a series of angles (same as pretest); 4) list all possible permutations of 4-colors (similar to pretest).

Part 1: Given a Figure, Write a Procedure

There were 4 variations of this type of question. Pupils were asked to write procedures that would produce a set of irregular stairs, a triangle, a rectangle and an angular figure. Answers were judged in terms of the correctness of 1) direction of angle; 2) size of angle; 3) length of line. Results are as follows:

Percentage Correct

	direction	size	length
Class B	79	79	79
Class A1	31	31	38
Class A2	47	67	80
Class B	75	08	71
Class A1	56	06	50
Class A2	73	07	67
Class B	80	75	79
Class A1	56	44	25
Class A2	73	73	27
Class B		17	54
Class A1		06	44
Class A2		06	25

Two interesting findings emerge from these results. One is the extent to which Class A2 resembles Class B in its percentage of correct responses. With little previous experience in measuring and rotation, a large percentage of Class A2 was able to give the correct values for the procedures. In contrast, the pupils in Class A1 showed a lack of understanding of many of the computer commands, despite the fact that they had used these figures often in their designs. Their responses to the first question, in particular, showed that a large number of children had no clear sense of turtle commands (e.g. FD 3, RT 5, LT 180, FD 4 ...). Secondly, the large percentage of errors in regard to size of angle on items 2 and 4 represents a problem inherent in the turtle language. Although each internal angle within an equilateral triangle is 60°, the turtle must be programmed to go around the exterior of the triangle. Its turn, then, is not 60° but 180° minus 60°. Few children realized this and perhaps would have come to an intuitive recognition only by actually walking around an equilateral triangle.

Part 2: Given a Procedure, Draw a Figure

This group of items was designed to measure pupils' ability to differentiate between different commands. If reflected the evaluator's concern that children were not aware of the

effects of the component parts of a procedure. The three procedures given were:

a) REPEAT 4 FD 40 RT 90 FD 10 LT 90
b) REPEAT 4 FD 40 RT 90 FD 40 LT 90
c) REPEAT 4 FD 40 RT 45 FD 40 LT 45

As in the case of the previous set, responses were judged in terms of the correctness of direction of angle; size of angle, and length of line, with the following results:

	Percentage Correct		
	direction	size	length
Class B	71	88	92
Class A1	25	69	25
Class A2	33	87	67
Class B	50	79	79
Class A1	13	63	19
Class A2	33	80	53
Class B	42	42	46
Class A1	06	13	31
Class A2	33	27	40

Most pupils recognized RT 90 as the command for a right angle; far fewer were successful with a 45° angle. Most of the attempts had no relation to the instructions given and seemed to indicate that the pupils had little idea of how to incorporate change in direction with an angle other than 90°. In fact, there were very few "near misses" on the last question. It seems that children either understood the gestalt or did not.

When correctness for angle size, direction and length were totaled throughout the questions as a whole, the following average scores were obtained:

	direction (1-6)	size (1-7)	length (1-7)
Class A1(n=18)	1.9	2.4	2.3
Class A2(n=18)	2.8	3.7	3.1
Class B(n=24)	4.0	3.9	4.9

The generally low scores of the pupils in Class A1 reflects the specificity of their learning. For example, although approximately 2/3 drew a right angle in response to RT 90, only half of those pupils were able to respond correctly to the inverse operation (i.e., a command of RT 90 in response to a right-angled figure). These children also showed little understanding of relative length, and their sense of direction was particularly poor in the interpretation of procedures. However, because these concepts were embedded in command fundtions, their confusion about the effect of the functions undoubtedly contributed to the results.

Part 3: Angles.

As noted previously, pupils in School B had been taught to recognize and draw angles of various sizes, while the children in School A had no experience with angles before the training program. During the program, the pupils in Class A2 had received some instruction from their teacher, and this appeared to affect their performance. On the post-test, 19% of Class A1 and 33% of Class A2 could draw a 90°angle. The same percentage in Class A2 could recognize a right angle in an unfamiliar tilted position, although only one pupil in the other class could do so. Not surprisingly, correct responses to the angles questions were closely correlated to the successful description of procedures. Of the five highest scores on the procedures questions, four were produced by pupils who recognized and drew 90° and 45° angles.

Part 4: Permutations.

A permutation question had been given on the pre-test, so it was possible to compare results. Success rate rose among the pupils in School A, with 6 giving the complete set of permuations (in contrast to 1 on the pre-test). A corresponding increase appeared at School B (from 2 to 11). All pupils who gave the entire set of permutations showed evidence of an orderly progression through the range of possible combinations.

Teachers' Appraisals

As the programs differed in the two schools, so did teachers' attitude toward it. The teacher of Class A1 thought that some children, particularly those who were receiving learning disability tutoring, got some positive feelings and a sense of accomplishment. However, she saw little carryover to their classroom work. Pupils saw it as an "isolated program." With few exceptions, their attitude and achievement in the program were predictable on the carryover of positive attitudes. Noting that some of the children who were not scholastically tops did very well on the computer, she saw this new confidence carried over into the class. She cited two Spanish-speaking boys who were below average in general academic work. On the computer, possibly unhampered by the problems of verbal communication, they seemed to flower. Working in a purposeful, competent way, they showed an aspect of themselves that could not have been predicted. Another boy, who was usually anxious and impulsive in the normal course of school work, found a new self-confidence. In general, she believed that pupils felt more positive.

Structure also concerned the class teacher at School B, but his criticisms were directed at the physical set-up rather than at the teaching methods. As discussed previously, the split of the class into two groups resulted in half the class remaining in the room doing project work while the others were at the computer. This caused a feeling of resentment among those who were scheduled to remain at their desks. They were interested in seeing what was happening and

found it hard to settle down. The idea developed that some had to do tedious work while others were having fun. (This was not strictly true, since each half was scheduled for daily computer time). In spite of this, their teacher was very enthusiastic about having the computers within the classroom. As a math specialist he could see their potential for mathematical learning, and he used the more enthusiastic of the 6th graders to teach Logo to the 7th and 8th grades.

Asked if he could see such a course as a permanent feature in the curriculum, he answered,"Definitely yes to turtle geometry." On the whole , the class felt great enthusiasm for the computers. "They couldn't get enough of it. It gave them something to look forward to. The things that went before were less of a chore." Fascination with the computer appeared to bring about a new seriousness of purpose to school work in general.

Tutors' Appraisal

The tutor organizer, who was himself a computer enthusiast, was more cautious in his appraisal. Indeed, he questioned a fundamental premise of the training, that computer programming could be used as an introduction to problem solving activities. He believed that, in the program, the children not only failed to reach the level of programming ability that would allow them to attempt problems, but they were presented with no strong incentive to develop the necessary attitudes. He attributed this to the "bang per buck" effect of Logo. Problems involving recursive designs and line drawings can be seen as tame and difficult when compared with the startling visual effects so easily achieved with random input and massive repeats. Milder, more esoteric goals that involved frustration, could be looked upon as a case of diminishing returns. There is little motivation to persist in the face of difficulty when immediate success can be obtained almost effortlessly.

He also suggested that there are certain features of Logo that are unneccessarily confusing. For example, in Logo both length and rotation are signified by a combination of letters (FD, RT, etc.) and numbers. There are no aids to distinguish between the two, although they signify qualitatively different operations. If symbolic representation were used as commands, children might be more aware of the fundamental differences between the two.

Pupils' Appraisals

The pupils were asked for their reactions to the program. These were generally favorable, leaving little doubt that they enjoyed it. Aside from specifics inherent in the two langauages (turtles vs. sprites), the comments from the two schools were similar and are amalgamated below.

What was most fun about learining to use the computer?

- You feel like an inventor.
- You find out things, like if you told the computer to go LEFT 20, you won't know what would come out.
- When I make a real neat procedure and people say---Oh! Can I have that procedure on my disk?
- When the turtle keeps going off the computer and makes funny lines.

In what ways do you feel smarter?

- When I make or learn something new by myself.
- Being able to do something that is pretty.
- Because I did the difficult one, the EXPLOTION.

Summary and Conclusions

What was accomplished? What could have been improved? Does Logo make a significant contribution to children's understanding of mathematics? There is no doubt that the schools differed in fundamental ways---the preparation of the children, the atmosphere to which they were accustomed, the expectations that teachers held for them, and the computer set-up itself. Conclusions that one might draw on the basis of one group were often contradicted by evidence from the other. Furthermore, the organization of the program, relying as it did on the enthusiasm of untrained tutors, could not be called ideal in any sense. However, some points emerge:

1. Work with Logo did affect some basic learning. For example, pupils improved in their ability to recognize and describe length and a basic angle (90^{o}). However, there is evidence that this learning was not generalized. For instance, although 69% of Class A1 (who had received no formal training in angles) were able to write a Logo procedure to tell the turtle to draw a 90^{o} angle, only 19% could draw it directly with pencil and paper. Furthermore, pupils who had not had previous experience in measuring a variety of lengths failed to notice changes in the FD command from one procedure to another. It seems that their understanding of these concepts was embedded in a particular context, namely turtle geometry.

2. Attitudes toward Logo material may be affected by previous learning experience. Children who had a history of independent work and explicit teaching of the math concepts involved in turtle geometry and sprites showed more sustained intellectual involvement than those who had not. On the other hand, some low achievers and children with reading disability also gained new confidence that was justified by performance.

3. Although the pupils enjoyed their work with computers, it is unclear whether or not they felt a sense of personal control in terms of being able to program the computer to do a variety of tasks or designs. For many children their "computer experience" was essentially affective and aesthetic in character rather than intellectual. This is not necessarily invalid; the disadvantage to this approach is its limitations. Since the computer is not essentially an aesthetic tool, in order to explore its visual capabilities one must engage in a cognitive effort. An understanding of the processes is an essential condition for exploration. Otherwise one is stuck at the affective level which, because it depends upon visual excitement, loses its appeal with repetition. By the last week, it seemed that the computer had lost its appeal for some children, and this appeared to be related to their lack of cognitive involvement.

References:

1. Papert, Seymour. Mindstorms. 1980. New York: Basic Books.

2. Howe, J., Ross, F., Johnson, K., Plane, F., Inglis, R. Learning Math Through LOGO Programming: The Transition from Labratory to Classroom. 1982. Department of Artificial Intelligence, University of Edinburgh.

EDUCATIONAL COMPUTING POST HASTE: A CASE STUDY

Deborah E. Blank

Electronic Learning Facilitators
9510 Linden Avenue, Bethesda, MD 20814

Enthusiasm for educational computing can lead to the purchase of computers before plans for their integration into the curriculum have been considered. This case study details the evolution of a computer education curriculum that was designed to accommodate two factors: 1) the computers were already in place, and 2) the principal of the school wanted every faculty member and student involved in hands-on computing as soon as possible.

INTRODUCTION

How quickly can an elementary school devise a plan for hands-on computer education for every student, particularly when the faculty is untrained and the computers have already arrived? An independent group of computer educators was asked in May, 1982, to design a kindergarten through eighth grade computer curriculum and prepare teachers to help students learn about computers. This paper will describe the curriculum that was produced, how a faculty was trained to teach it, and how its implementation is proceeding.

The setting is a parochial school located in a medium-sized northeastern city. Most of the 500 students come from middle to upper-middle class homes. The principal is totally committed to computer literacy for all of her students, and is bolstered by parental support that guarantees that all wishes will come true.

The school's board of directors had purchased eight Apple II Plus microcomputer systems; two of them were being used occasionally to run software, and the remaining six were sitting idle. Another component of this idyllic situation was a faculty that was eager to learn about computing - even if it meant learning along with, or from, their students.

The principal set these requirements for the computer curriculum:

1) All students (K-8) had to have hands-on computer time for at least half of each school year.

2) The computer lab would be staffed by regular faculty, and computer literacy activities would be carried out in every classroom.

3) Students were to use computers for as many purposes as possible.

4) The curriculum would be implemented in September.

DEVELOPING THE CURRICULUM

Design of the curriculum proceeded according to these basic principles:

1) Knowledge of the cognitive development of children must be reflected in the curriculum goals and activities.

2) Computers are tools, not a subject to be studied, and should be integrated into the existing curriculum.

3) Computer programming is a valuable activity for children because it encourages systematic problem-solving and logical thinking.

The next step in designing the curriculum was to identify goals for the entire project. These are:

1) To help students, teachers, and staff develop a sense of control over computers.

2) To help students, teachers, and staff discover computer applications that are useful to them.

3) To encourage students to develop an appreciation of computers as tools for life-long learning.

4) To encourage the development of problem-solving skills that can be applied to all facets of living.

5) To stimulate thought and discussion among students, teachers, and staff about the advantages and disadvantages of computer use in school and in society.

6) To generate interest in the field of computer science and related careers.

Once these goals were established, the concepts that were to be the basis for the classroom and lab activities were

defined. The curriculum was divided into two parts: Computer Literacy and Problem-solving. The programming language LOGO was selected for grades kindergarten through four, and BASIC for grades five through eight. Concepts were written for each grade level for both sections; many of the concepts were repeated several times throughout the total structure so that knowledge and skills would become increasingly sophisticated as students progressed. Samples of the concepts are listed below:

Computer Literacy Concepts:

Level K: Computers can do things over and over again without getting tired.

Level 1: People need to share computers.

Level 2: Programs are sets of instructions that tell the computer what to do.

Level 3: People can learn the same things in different ways; learning from a computer is one way.

Level 4: Vocabulary to describe computer parts is extensive and constantly growing.

Level 5: Word processing requires the use of a program.

Level 6: Commercial software can be improved by thoughtful development and programming.

Level 7: Many electronic games are single-purpose computers.

Level 8: Computers are significantly altering the way we live.

Problem-solving Concepts:

Level K: Computers follow instructions given by people.

Level 1: People must use special words to give instructions to computers.

Level 2: Programs are made up of one or more procedures.

Level 3: A computer screen is a grid - all drawings are actually made by connecting straight lines.

Level 4: Some programming languages can be used to write interactive programs.

Level 5: Each programming language has a special vocabulary and syntax.

Level 6: BASIC is a highly interactive language that is user-friendly and good for writing educational software.

Level 7: Problems to be solved using computers can be broken into smaller subproblems that operate under the control of the main program.

Level 8: Solutions to computer problems are arrived at using algorithms.

Classroom and computer lab activities had to be planned to correspond with concepts. Because teachers would have a minimal amount of training before they began instructing students, each activity was written as a complete lesson plan. An important feature of each plan was the identification of skills from other subject areas that are practiced during the activity. These listings helped alleviate concern that other subjects were not receiving sufficient attention because of the new emphasis upon computer education. The goals, concepts, and lesson plans were combined into a curriculum guide that was provided for each teacher.

SELECTING SOFTWARE

Since the school had a rather loosely organized general curriculum, courseware selected could not be made to match specific objectives. Teachers requested the purchase of math drill and practice programs and some educational games. The remaining software was selected to provide examples of the varied uses of computers for instructional purposes, and to serve as the basis for some critical thinking activities (such as software evaluation) by students in the upper grades. A word processing program and a typing tutorial were purchased for use by the upper grades, and the LOGO language for the younger children. It was suggested that teachers select their own courseware as the year progressed, since the full software budget had not been spent. In addition, several magazines were ordered, and an assortment of books to provide technical information and ideas for using computers in the school.

SCHEDULING

Rather than expecting newly trained teachers to program in both LOGO and BASIC, the decision was made to teach LOGO to the younger children in the fall, and devote the spring to teaching BASIC to the older students. Each kindergarten through fourth grade class was scheduled to spend 45 minutes per week in the lab with a maximum of two children per computer. Fridays were reserved for lab use by students who needed to use drill and practice programs or for enrichment. All students were to participate in classroom computer literacy activities throughout the school year.

Both the principal and the teachers agreed that more computers would allow for year-round use by all students, and that their purchase should be a priority budget item. Because the principal felt that she could not afford to employ a full-time computer lab teacher, she decided that the lab would be staffed by faculty members who would spend one hour per day teaching hands-on computer use. The remaining portion of their school day would be spent with their regular classes. This meant that the lab teachers, as a

rule, were not instructing their own students in the lab.

STAFF DEVELOPMENT

The faculty returned to school a few days prior to the opening of the fall semester. Every teacher was expected to participate in two days of computer training. All but a few were enthusiastic, and all were somewhat intimidated by the equipment and scope of the curriculum. Two faculty members had had computers in their classrooms for several months, and had begun learning some BASIC. The remaining 23 teachers had no experience with computers.

Four trainers worked with 25 teachers, four administrators, and several parents for two seven-hour days. Eight of the teachers had been selected to staff the lab, including the two who knew some BASIC. Since these teachers would soon be teaching programming one hour per day, four days per week, they spent almost two full days with one trainer learning LOGO. These eight teachers became known as the "Intensive 8," and met for short periods with the larger group to be introduced to the curriculum guide and to preview software. The remaining teachers spent their days as follows:

- 1/2 day learning terminology and how to operate the computer to run courseware
- 1/2 day previewing and evaluating courseware
- 1/2 day (primary teachers) learning LOGO in the lab (taught by the "Intensive 8")
- 1/2 day (upper grade teachers) learning BASIC and word processing
- 1/2 day planning "what do I do on Monday?" - reviewing those portions of the guide that they would begin teaching the next week

By the end of the second day, both teachers and trainers were tired but satisfied. Several teachers remarked that the specificity of the activities in the guide would help "pull them through" the first weeks of teaching.

INFORMING PARENTS

During the evening following the first day of training, parents were invited to interact with the computers and learn about the new computer curriculum. Many parents came, and many stood with arms folded while a courageous few played educational games and moved the LOGO turtle around the screen. The questions that were asked after the presentation of the curriculum focused on the type of home computer to purchase to supplement the school's curriculum, and upon the widespread concern that time spent learning about computers would, by necessity, mean less time learning about other subjects.

EVALUATING THE CURRICULUM

During early October, approximately one month after the curriculum was implemented, one of the trainers visited the school for an informal meeting with the "Intensive 8." At that point there was both good and bad news. The good news was that the teachers had exceeded their own expectations and were able to instruct LOGO with a fair degree of confidence. The children were very excited, and were progressing rapidly. The unhappy news was that the schedule that called for a different teacher in the lab each hour was a disaster. There were two major problems: the teachers had no time to prepare for their programming classes (their one-hour preparation time was used to plan for their regular subjects), and their arrival at the lab was always rushed and disorienting. (Several parent volunteers were assisting in the lab, but their schedules were erratic.) Teachers were also concerned about having to teach BASIC in January when they were just mastering LOGO.

To help address these problems, the principal agreed to:

1) Hire (on a tuition-for-services-rendered basis) one of the parent volunteers who had a programming background.

2) Provide training in BASIC prior to the Christmas break, and allow teachers to take the computers home during that period.

Two complete, formal evaluations will be done during the next six months. The focus of the evaluations will be upon the proficiency of the teachers, and the value of the curriculum guide in helping them achieve their computer education goals. The results of these evaluations will be described at the NECC conference.

LOGO

Carolyn Markuson
Joyce Tobias
Martin Saltz
James Gottlieb
Bobbie Gibson
Roy Moxley
Steve Tipps
Hal Evans
Glen Bull
Terry Schwartz
Mary King
Steve Taylor
Susan Walker
Pete Davidson
Leah Rampy
Rochelle Swensson
Barbara S. Hilberg

ABSTRACT: LOGO - A Three Year Sequence, Grades 4-5-6

Carolyn Markuson, Joyce Tobias, The Public Schools of Brookline, Brookline, MA 02146

Seymour Papert created a learning environment, where learning becomes as natural a process as walking and talking, all with a new technology: the microcomputer. Why is LOGO such a unique language for elemantary grade children? What characteristics of LOGO enhance the total learning environment - challenging and honing the problem solving capabilities of young children - teaching logical thinking skills and creativity at the same time - encourages children to test their ideas and receive immediate response? How can spatial and number relationships, as well as geometric proportions, become part of a child's consciousness?

LOGO has been taught in Brookline schools since the 1978 pilot program sponsored by MIT and the National Science Foundation. Today, it has evolved into a three-year sequence for all children in grades 4, 5, and 6. The curriculum developed to support this program, the in-service training program developed for classroom teachers, and the decisions required to select a version of LOGO best suited to the needs of this year-long educational experience will be presented.

ABSTRACT: Development of a Program Designed to Use LOGO and a Floor Turtle in a Nursery School Environment: Trials, Tribulations, and Triumphs

Martin Saltz, James Gottlieb, Bobbie Gibson, Roy Moxley, West Virginia University, Morgantown, WV 26506

With some knowledge of microcomputers in the schools and funds available to buy the needed equipment, the Director of the West Virginia Nursery School (College of Human Resources and Education) decided to develop a program introducing LOGO and a Floor Turtle to a group of nursery school youngsters. The Director and others at the College of Human Resources and Education became involved in developing a curriculum.

This presentation is designed to:

1) discuss the process used by a group of educators while developing a school program involving LOGO and Terrapin Turtle in a nursery school environment;
2) present an account of the program as used by teachers and children in the classroom;
3) report on the behavior of adults and children as the program was developed and used; and
4) evaluate and suggest modifications to the program based upon observations made during the developmental phase.

The two presenters were among those initially involved in the developmental phase of this course. They will present the basic information through a lecture. This lecture will be augmented by slides which will show faculty, staff, children, and parents involved in the project. Audience interaction will be invited and an annotated bibliography of reference and research material used during the course development will be available.

ABSTRACT: LOGO Instructional Development Project

Steve Tipps, Hal Evans, Glen Bull, University of Virginia School of Education, 405 Emmet St., Charlottesville, VA 22901, Terry Schwartz, Mary King, Steve Taylor, Susan Walker, Pete Davidson, Albemarle County Schools

The University of Virgina School of Education and Albemarle County Schools joined in a cooperative effort to develop and monitor LOGO in fifteen fourth grade classrooms. The major components of the project were:

In-service training--conducted by two University professors over the full year of the project. The goal of inservice was proficiency with LOGO.

Implementation--carried out by the teachers with records of progress and change in the actual classroom. Journals and weekly reports were kept.

Evaluation--cooperatively done with both standardized testing for possible cognitive and attitude change and specific testing for LOGO proficiency.

The project began in the summer of 1982 and has gone throughout the year. Teacher had two months of training before taking the computers into the class. In-service continued throughout the year with teachers discussing implementation questions and learning new language skills as they worked on programming projects.

Testing was done early with the children involved in the project. Specific instruments were the Cognitive Abilities Test, the Fennema Mathematics Attitude Test, and Test of Intrinsic-Extrinsic Classroom Motivation. Post tests will consist of the same tests and exercises in LOGO and problem solving.

From the records and experience of teachers in this year's project, instructional suggestions will be compiled for expansion of the project into fifth grade.

ABSTRACT: The Programming Styles of Fifth Graders Using LOGO

Leah Rampy, Rochelle Swensson, School of Education #337, Indiana University, Third & Jordan, Bloomington, IN 47405

Twelve fifth grade students from a local elementary school participated in a six-week long class at Indiana University designed primarily 1) to introduce students to the LOGO language and 2) to study the programming styles exhibited by students learning LOGO. The classes involved large group instruction, time to work on the computer on instructor-suggested as well as student-initiated projects, and opportunities for students to demonstrate their programs and to exchange ideas. One Apple II-plus microcomputer was available for every student. Undergraduate elementary education majors, trained in LOGO, served as tutor-observers for the students. The observations suggested similarities and differences among the participating students' approaches to programming.

A secondary concern of this study was to examine the effect of cognitive style on programming style. Researchers in the Brookline LOGO Project (1979) had argued that students' cognitive styles were reflected in their approach to programming but apparently no attempt was made to measure the cognitive style of the students involved in that project. In this project, students were selected on the basis of extreme scores on the Children's Embedded Figures Test, a measure of field dependence-independence.

Because of the growing concern about a possible gender gap in computer literacy, a final focus of this study was a comparison and contrast of the programming styles of boys and girls.

The NECC presentation will present a brief summary of the findings of this project with regards to 1) the programming styles of fifth graders using LOGO; 2) the relationship of field dependence-independence to programming styles; and 3) the relationship of gender to programming styles.

ABSTRACT: Modifying Papert's Vision: LOGO Lessons

Barbara S. Hilberg, Electronic Learning Facilitators, 9510 Linden Avenue, Bethesda, Maryland 20814

Seymour Papert envisions children discovering LOGO in a total LOGO environment, which is freely accessible to them throughout the day. While this is the ideal, very few schools at present have both the commitment and the funds necessary to provide this. A reasonable alternative is the LOGO class - held for a specified time - either within a school program or as an after school activity.

The project presented here describes a series of LOGO classes, offered by Electronic Learning Facilitators to children, ages 5 through 12, both during the summer and after school during the school year. The success of the program was dependent on two factors: 1) the format, which provided for a balance of structure and discovery; and 2) supplemental materials, which allowed children without computers at home to continue to explore LOGO concepts outside the classroom setting.

The children were divided roughly into younger and older groups, working two to a computer. All groups in the introductory class met for a total of 10 hours, the summer groups meeting on consecutive days, the fall and winter groups once a week. Follow-up classes were also available.

Because of time limitations, teachers made brief presentations of new concepts to the class followed by time for children to experiment individually. A series of verbal, physical, and spatial relationship activities preceded and accompanied computer use. There was a strong emphasis on writing procedures and problem solving. Supplemental materials, such as Program Puzzlers and Playing Turtle, were developed for distribution to children to be used outside of class. These and other materials developed for the program will be described and demonstrated.

ALTERNATIVE APPROACHES TO PROVIDING COMPUTING FACILITIES

Dr. Mary Lucy Sennett
Richard V. Murdach
Pat Kelly
Richard W. Evans
Mary M. De Boer

ABSTRACT: CompuShare: A School-Community Project

Dr. Mary Lucy Sennett, Deer Creek School, Box 2086, Arcola, MS 38722

Over the past few years the message "no funds available" has echoed through the halls of all levels of academia. Unfortunately, the trends seem to predict an even more dismal financial future. In the midst of this period of financial chaos, we are also faced with volumes written on the significant difference microcomputers can make in education. However, there is one simple fact that must be faced. It takes money to purchase microcomputers. If the funds are not available, the current literature becomes lost rhetoric.

Project CompuShare is a program designed to develop a cooperative venture between small rural schools and small local business enterprises. The major objective of the project is to obtain free microcomputers for the schools from donations from small local businesses. The businesses involved will also realize both short- and long-range benfits from participating in the project.

The priorities of Project CompuShare include:

(1) To establish a work-study program where a member of CompuShare would pay a student minimum wages to do comuter work for their business. The work could be done on the computer that the business donated to the school.
(2) To develop an overall training program for students in the use of microcomputers in business applications.
(3) To provide an interchange of ideas and support between the school and its local business community.

The presenter will discuss guidelines of the project and will elaborate in detail on the benefits and problems that result for both the school and business by participation in Project CompuShare.

ABSTRACT: The Central Illinois Computing Consortium

Richard V. Murdach, Director, Central Illinois Computing Consortium, 1444 Maine Street, Quincy, IL 62301

During the 1981-82 school year an overwhelming majority of school districts in central Illinois identified the use of microcomputers for instructional and administrative applications as their number one need. The Western Illinois Center for Educational Improvement served as a coordinating agency to provide districts resources to serve the growing need for microcomputer information and in-service. It soon became apparent that individual districts would not be able to solve this problem independently without wasteful duplication of efforts and financial resources. However, if a number of school districts were to pool their collective resources, these needs could met be more effectively at less cost. Through the WICEI, a committee was formed to gather data and information concerning ways in which districts could collectively meet their individual needs for microcomputer support. The result of that effort is the formation of the Central Illinois Computing Consortium.

The Central Illinois Computing Consortium is designed to provide vitally needed educational services and technical assistance to school districts, nonpublic schools, vocational and special education cooperatives, community colleges, and other educational agencies in the fourteen-county region of the Western Illinois Center for Educational Improvement. The computer consortium will provide districts with in-service and staff development opportunities, technical assistance, software evaluation, membership in the Minnesota Educational Computing Consortium, collective bidding for hardware ands software, and access to a library of educational software in a highly efficient and cost-effective manner. Any or all of these services wiould be available to districts who join the Central Illinois Computing Consortium.

The presentation will concentrate on the benefits, development, and organization of an educational comuting consortium.

ABSTRACT: A Relocatable Computer Laboratory

Pat Kelly, Computer Resource Teacher/Coordinator, Carroll County Board of Education, Westminster, MD 21157

Federal funds have made it possible for Carroll County to establish the position of Computer Resource Teacher/Coordinator in order to create a Relocatable Computer Laboratory. The goal is to travel around to every Middle school in the county and provide some "hands-on" experience for each seventh and eighth grade student. It is one way of beginning to meet the rapidly developing need to provide "computer literacy" to our

students. In addition, instruction is provided for teachers and administrators in special workshops.

While this may not be the ideal solution to meeting this need, it offers an alternative that helps close the gap between the need to raise computer literacy levels and the high cost of purchasing sufficient hardware to do so. Many unanticipated difficulties were encountered in preparing schedules and purchase orders, purchasing hardware, and developing the curriculum. The presentation will share with others what has been learned, to assist them in avoiding pitfalls and oversights.

ABSTRACT: CALL: A Multipurpose Educational Computer Facility

Richard W. Evans, Associate Director, The Learning Center, SUNY College at Farmingdale, Farmingdale, NY 11735

The Computer-Aided Learning Laboratory (CALL) at the State University of New York College at Farmingdale is emerging as a very cost effective support mechanism for students and faculty at the college. CALL utilizes twelve Apple II+ microcomputers linked with a ten megabyte Corvus hard disk and Constellation network. This advanced hardware configuration provides educational computing support to the college with great flexibility at low staffing and maintenance costs. The staff has created a turn-key instructional program which allows students new to the facility to quickly learn its operation and gain access to its courseware. Since the ten megabyte disk holds the equivalent of 64 mini diskettes of software, there is no need to maintain and handle a library of floppy disks. This allows the facility to maintain minimal staffing - generally a workstudy student to maintain records and assist students in the selection of courseware. In addition to reducing staffing costs, the network configuration also reduces maintenance costs: the weakest links in our hardware configuration are the floppy drives required to mount courseware on the storage disk. The ability to avoid the use of floppy drives for student use has made it possible to operate all twelve microcomputer stations with zero downtime since the assumption of full-time operations in Spring, 1982.

The facility offers a wide variety of educational computer support to the campus. As part of the College's Learning Center, CALL provides students with both basic skills support (reading, writing, and mathematics) and tutorial assistance in content areas. Courseware is available in Physics, Chemistry, Biology, Sociology, Psychology, Mathematics, Secretarial Science, and Spanish, as well as in basic English structure, reading skills, and mathematics. Student use of the facility for these purposes has grown dramatically since the facility began operation. A continuing program of evaluation reveals that students react very favorably to the important aspects of CALL's operation such as its General Functioning, Potential, Proctoring Assistance, and the Interest and Effectiveness of its courseware (these results are all highly significant with $p<.001$).

In addition to students' independent use of CALL for basic skills and tutorial purposes, the facility operates in a number of classroom modes to support instruction. Special classes in Physics, Spanish, and English composition (utilizing the APPLEWRITER word processing program) make organized use of the facility with very positive results. It is anticipated that this use will soon expand to include laboratory exercises in Chemistry, Biology, and Psychology. Another role played by the facility is that of providing computer literacy workshops to area educators and to the college's faculty. Workshops as well as individual visitations are continually held, and it is expected that the facility will have a generative effect in increasing the college's use of educational computing.

ABSTRACT: Cost Effective Implementation of a Microcomputer Program in the Elementary School

Mary M. De Boer, Assistant Director, The Learning Center, SUNY at Farmingdale, Farmingdale, NY 11735

Seven alternative microcomputer implementation plans for a New England School District were examined to determine the most cost effective approach. Areas of major concern in choosing a program were centralized vs. decentralized labs, curriculum specific vs. general CAI software, and age/grade level of students. The key factors used to determine overall program cost were training time, hiring of new staff, hardware, and any additional costs (such as busing) resulting from plan requirements.

This particular school district had 13 elementary schools with a total enrollment of 5,583. They had made a recent purchase of 20 microcomputers and had available some limited software programs but had no implementation plan. In addition they were greatly affected by budget cutbacks yet wanted to provide a quality educational program utilizing the microcomputers for as many of their students as they could.

The following seven plans with their projected costs, exclusive of hardware costs, were proposed and examined:

PLAN 1: To place microcomputers in the Resource Rooms, implement computer literacy/general CAI for grades 4-8, 9 weeks, 4 cycles/year
costs: yr 1 - $18,900, yr 2 - $17,500

PLAN 2: Place microcomputers in classrooms, scattered throughout the schools at various grade levels
costs: yr 1 - $19,200, yr 2 - $17,500

PLAN 3: Set up one or more centralized computer labs providing general CAI
costs: 1 lab, yr 1 - $34,000, yr 2 - $33,500; 2 labs, yr 1 - $60,000, yr 2 - $59,500

PLAN 4: Place microcomputers in libraries. This plan was immediately rejected since the costs were enormously prohibitive, as the school district had no library staff members.

PLAN 5: Implement curriculum relevant CAI programs with microcomputers in the classrooms covering various grades
costs: yr 1 - $35,000, yr 2 - $30,500

PLAN 6: Curriculum relevant CAI with microcomputers in centralized labs
costs:1 lab, yr 1 - $34,500, yr 2 - $33,500; 2 labs, yr 1 - $60,520, yr 2 - $59,520

PLAN 7: Curriculum relevant CAI with microcomputers in the Resource Rooms
costs: yr 1 - $35,000, yr 2 - $30,500

General CAI and computer literacy were chosen over a curriculum relevant approach because of the history of difficult and unsuccessful implementation of curriculum relevant CAI in elementary schools. Plan 1 was chosen because it provided quality computer literacy/general CAI exposure to a large number of students, while it did not strain the school district's budget. The higher cost in year one was incurred due to training time of staff during the initial implementation phase; year two includes maintenance costs.The grades 4-8 were chosen on the basis of a minimum of nine hours exposure for each student, with a priority for older students to better aid in their preparation for high school.

DISTANCE TEACHING OF SOFTWARE ENGINEERING

Darrel Ince
The Open University
United Kingdom

W. S. Matheson
The Open University
United Kingdom

As part of a major programme of scientific and technological updating, the Open University is developing a novel Masters Course which deals with the industrial applications of computers. The course is intended for practising programmers, engineers and technical managers who work in a real-time environment and who are finding their work transformed by the micro-computer.

The course is intended to alleviate a major problem which is currently facing British industry over the retraining of existing personnel. There are now a large number of personnel working in areas such as process control, avionics and command and control systems, who require updating on the software and hardware aspects of microcomputer systems. Unfortunately, these personnel often occupy critical positions in their company and cannot be released for the 12-18 month period necessary for the study of a conventional Master degree. We hope that the Open University, an institution set up to provide degree-level education for the house-based student, will be able to respond to this challenge, and provide a model which can be used in other shortage areas where there is a need for scientific and technological updating.

The first part of the paper will describe the work of the Open University. The background to the course will also be described and the structure and aims of the course outlined.

The second part of the paper will describe one of the major modules of the course, software engineering. This module, which will be taken by the majority of students who enter the course, takes the view that coding represents only a small part of the program development process, and that other activities (analysis, specification, design, maintenance) are, at least, equally important. The module will consist of a number of components:

(i) Course texts which deal with parts of the software life cycle.

(ii) A number of industrial, real-time software case studies.

(iii) A home experiment kit which consists of a stand alone micro-computer which can act as a terminal to the Open University mainframe network, for 'hands-on' experimental work.

(iv) a video tape which shows the progress of an actual software project using a large number of personnel.

The paper will conclude with a discussion of some of the difficulties encountered in providing such a module to the postgraduate student studying at a distance.

District Planning for Computer Use in K-12

Glenn Fisher
Alameda County Superintendent of Schools Office
Hayward, CA 94541

ABSTRACT

An outline for a K-12 computer literacy scope and sequence model will be discussed by Gary Bitter. Included will be computer awareness and computer programming details. A brief implementation model for curriculum will be discussed.

Don Rawitsch will discuss the following:

1) In creating an instructional computing plan, instructional expertise is more important that computing expertise.

2) Although implementation factors such as equipment, materials, and training take up the most time in the planning process, these items should not be discussed until a rationale is developed for why the district thinks it should be involved in computing.

3) The district must determine the relationship of computing to its curriculum, considering both how computing can be integrated into the present curriculum, and what new curriculum areas might be suggested by computing.

4) Computer literacy is much more effectively developed in students when it is planned as a cumulative effect of activities throughout the curriculum, as opposed to being set aside as a single course.

5) The purchase of computing equipment should be determined based on educational goals more than by hardware features and cost.

A variety of questions to consider will be addressed by Pristen Bird.

Among these are:

What do we want to do with computers?
. For which students? staff?
. What courseware is appropriate?
. Do we need to develop our own software?
. Can we afford to?
. Which brand(s) should we buy?
. What monies will be committed?
. How will we train staff?

The chair will discuss the role of the administrator in this process. District planning is a political process and requires support of staff, Board, administration, and parents. It should parallel other curriculum planning in the district. Staff development is an important part of a plan.

The plan should extend over time to demonstrate commitment and the possibility of change and input. There should be room for serendipity and learning at school sites within the district plan.

The administrator is the crucial element in successful implementation.

PARTICIPANTS:

Gary Bitter
Arizona State University
Tempe, AZ 85287

Pristen Bird
Instructional Computing Consultant
Department of Education
Tallahassee, FL 32301

Don Rawitsch
Director of User Service, MECC
St. Paul, MN 55113

Information Technology and Its Impact on the United States - Overview and Implications

Sponsor: ICCE

Linda Garcia
Fredrick Weingarten
Office of Technology Assessment
U. S. Congress
Washington, D.C. 20510

Linda Roberts
Office of Library and Learning Technologies
U. S. Department of Education
Washington, D.C. 20202

The "information revolution" is profoundly affecting American education and training -- creating new demands for instructional services and, at the same time, providing new opportunities for improvement and delivery of such services. The new information technologies can help all educational institutions meet these new demands. Many are already being effectively used in education and training. However, OTA has identified a number of barriers to their use -- their high initial cost, the lack of high quality programming, and the shortage of local personnel with adequate training. Whether or not new information technologies will fulfill their potential will depend, in part, on the kinds of actions that the Federal Government takes. What is needed is a broad approach that takes into account the changing needs for education and training, considerations of equity, and changing institutional roles.

THE ELECTRONIC BLACKBOARD

USING A MICROCOMPUTER AND LARGE-SCREEN TELEVISION AS A LECTURE AID

JAMES E. CLARK
Department of Economics
Wichita State University
Wichita, Kansas 67208
Telephone 316-689-3220

ABSTRACT

This paper will describe and demonstrate the development, use, and benefits of programs that use an Apple microcomputer and large-screen projector television as the primary lecture aid (replacing the traditional chalkboard or overhead projector) in large lecture sections of college Principles of Economics classes. Some of the benefits of using the Electronic Blackboard are: 1) its attractiveness to students; 2) its ability to interweave text with graphics and simulations; 3) its legibility, even in the back of large (250+ seat) classrooms; 4) its ability to use the Apple's high-resolution colors to highlight and tie together key concepts. The Electronic Blackboard can be used to prepare lecture materials before class, and also can be used in a "live" mode to display text and graphics created during class; examples of both types of presentations will be shown.

ORIGINS OF PROGRAM

The impetus for the Electronic Blackboard program came from a National Science Foundation CAUSE (Comprehensive Assistance to Undergraduate Science Education) grant entitled "Interactive Microcomputing in the Classroom." The purpose of the grant is to encourage the use of microcomputers in classroom settings for creating demonstrations, simulations, etc., that can be easily manipulated to answer "what-if" questions from students. Funds from the grant have been used to purchase 19 Apple II+ microcomputers and seven Kloss Novabeam projector televisions with large (4' x 6') screens. On a competitive basis, 22 faculty members at Wichita State University have been selected to receive summer stipends for the development of software to implement the objectives of the grant. To date, the following disciplines have been involved, either through summer sub-grants or voluntary participation: astronomy, biology, chemistry, economics, electrical engineering, industrial education, industrial engineering, mathematics, and physics.

Many excellent modules have been produced by participants in this grant, ranging from simulations of queuing and other random processes to the graphical depiction of the evolution of the universe; some of these programs were demonstrated at NECC 1982.[1,2] While these modules, and similar ones created elsewhere, have been valuable to students,[3] they have not made full use of the microcomputer's ability to enhance instruction, particularly in the large-section classroom. The Electronic Blackboard is a step further in integrating the microcomputer into the classroom.

ADVANTAGES OF THE ELECTRONIC BLACKBOARD

Compared to chalkboards and overhead projectors (the traditional lecture aids for large classes), the Electronic Blackboard has several advantages that make using it worth the necessary investment in equipment and preparation time. Among these are:

Attractiveness to students

The current generation of students are accustomed by a lifetime of experience to watching the movement and color of television shows; the black-and-white, static pictures produced by chalkboards and overhead projectors are, by contrast, very boring and unexciting. The Electronic Blackboard can promote student attentiveness and learning by providing the color and motion to which students have become accustomed and attracted. In addition, since the Electronic Blackboard embodies up-to-date technology and resembles (faintly) an arcade game, students are more likely to pay attention in class to material presented on the Electronic Blackboard.

Legibility

When properly prepared, text and graphics presented on the Electronic Blackboard are much more legible to students than are the same materials written on a chalkboard or overhead projector. This is especially true for students who are forced by large class sections to sit in the far reaches of several-hundred-seat lecture rooms. Proper presentation does require that attention be given to clear and unconfusing screen layouts (this is also necessary for clear presentations on chalkboards and overhead projectors). To insure legibility, it is also necessary to use non-standard character sets. The Electronic Blackboard uses both modified, all-white, standard-size upper and lower case characters, as well as slightly larger upper and lower case letters with true descenders; this larger character set greatly improves legibility, especially in large classrooms. Details on the character sets used may be found in the section below on Program Description and Capabilities.

Color

The ability of the Apple microcomputer to create colored lines and images has been harnessed in the Electronic Blackboard to make presentations both clearer and more interesting. In particular, complex presentations involving graphical analysis (common in economics, and in many other disciplines) can be made much clearer, and associated elements tied together, with a careful use of color. While colored diagrams can be created on chalkboards and overhead projectors, such diagrams are typically quite messy, and very difficult to modify to show the effects of changes.

Animation

The one area where the Electronic Blackboard represents the greatest improvement over its traditional counterparts is the microcomputer's ability to manipulate rapidly large amounts of information, and to clearly display the results. While this ability is most obviously applied to purely numerical information, it can also be very effectively applied in classrooms where the relevant information concerns the location of a line on a graph, or the position of an image on the screen. The microcomputer can be easily used to create, display, and manipulate numerical and graphical examples, simulations, and so on, in ways and at speeds that are simply not possible with the traditional tools. This is especially true in cases where students ask questions which need substantial calculations to answer. The Electronic Blackboard is designed to allow easy insertion of modules representing graphical and numerical examples, simulations, and other similar types of materials; several examples of such modules will be presented, along with examples demonstrating the other advantages of the Electronic Blackboard.

STUDENT REACTION TO THE ELECTRONIC BLACKBOARD

The author has used the Electronic Blackboard as the primary lecture aid in teaching two large class sections of Principles of Economics during both the Fall 1982 and Spring 1983 semesters at Wichita State University. While final course evaluations are not yet back, some preliminary evaluations are available. Student attitudes toward the use of the Electronic Blackboard, as expressed both in class discussions and in outside conversations with the author and other faculty members, have been almost unanimously favorable. Comments show that students feel that text materials presented on the Electronic Blackboard are more interesting and easier to learn from than would be the case if the same materials were presented using traditional methods; the reaction to graphical materials has been even more favorable, with stress placed on the increased understanding that is generated by the used of color and motion. In addition to classroom use, demonstrations of materials prepared on the Electronic Blackboard have been presented to conferences of persons interested in computer-based education[4] and to economic educators[5], with highly enthusiastic and positive receptions.

Along with its benefits, several problems have arisen with the use of the Electronic Blackboard. Due to the design of the screen, the intensity of the image from the projector is diminished when the screen is viewed from a large angle; with the present equipment, the Electronic Blackboard requires several projectors and screens for classrooms that are wide relative to their depth. Also, prepared text materials appear on the Electronic Blackboard much faster than they can be written on a chalkboard or overhead projector, creating a tendency for the instructor to present materials faster than students' note-taking can keep up.

For lecture materials prepared in advance of classes, creating materials for the Electronic Blackboard takes several times as long as does preparing the same material for writing on a chalkboard or overhead projector. At present, preparing one class hour's worth of material takes between two and three hours of time for the Electronic Blackboard, with considerably more time necessary for complex graphics; this compares to the approximately half-hour usually required to prepare an hour's worth of chalkboard presentation. The benefits of the Electronic Blackboard seem (to the author) to be worth the time involved, especially since the materials created can be reused in future classes, with any needed modifications made fairly rapidly. An authoring system for the Electronic Blackboard is at present being developed that should substantially reduce preparation time.

PROGRAM DESCRIPTION AND CAPABILITIES

The core program of the Electronic Blackboard is written in Basic and is based on the concept of organizing the material to be presented into "pages;" one page is normally one full screen of material. Within each page, the program writes a section of text onto the screen, then waits for the space bar (or firing button on paddles or joystick) to be pressed before writing the next section of text. Section length can be anywhere from one character to an entire page, and can be varied for each section. "Turning pages" (clearing the screen and starting a new section of text) is also accomplished by pressing the space bar. The instructor is thus always in control of how fast the material is presented. If desired, it is possible to have more than one screenful of text on a particular "page;" the screen can be scrolled to make room for new text without erasing the entire screen. It is also possible to go back to previous pages or to move forward by skipping pages at any time.

Text and graphics are displayed on the Apple's high-resolution screen through the use of Image Printer (a machine-language graphics program soon to be available from C & C Software). As used in the Electronic Blackboard, Image Printer can very rapidly write upper and lower case letters, numbers, and symbols anywhere on the Apple's high-resolution screen (regardless of byte boundaries). Character sets available include a standard-size (7x8 dots) all-white character set that is much clearer than the Apple's standard character set, an enlarged (9x12 dots) white character set with true descenders, and a small (6x6 dots) character set with only upper case letters and numbers. Other character sets in other sizes can be created if the user desires. For use in

large classrooms, legibility is best with the large character set, although the standard-size characters can also be used, and are appropriate for tables, labels, and so on.

The maximum amount of material contained in an Electronic Blackboard program is limited by the amount of memory available. The Image Printer routines and character sets fill the space below the Apple's high-resolution graphics page 1; if both high-resolution graphics pages are to be used, the space available for programs is from 24576 to 39590 for a 48K Apple with MAXFILES1 set. This is enough space for 15-20 average pages of text with colored underlining. Longer presentations can be created by having one program end by running the next program. This can be done without disturbing the text or graphics being displayed, so that the instructor can be explaining the present display while the next program is loading.

The organization of the program into "pages" makes it easy to add modules of graphics displays, simulations, etc., into the program (Image Printer is also an excellent way to create graphics displays and animation). Each page begins with an even-thousand line number (page 1 starts at line 1000, page 13 starts at line 13000, etc.), so that by appropriate numbering a module can be executed at the appropriate time in the program. Modules created outside the Electronic Blackboard can be renumbered and merged into the Electronic Blackboard using any good commercial program editor.

Graphics and text can be very effectively interwoven by putting graphics on one of the Apple's high-resolution pages and the corresponding text on the other high-resolution page. At any time, the Electronic Blackboard can flip between the two pages by pressing the "1" and "2" keys.

Also available is a nondestructive pointer (an arrow shape) that can be placed on the screen to point to key terms and illustrations. The position of the arrow can be controlled from the keyboard, or by a joystick or paddles.

The Electronic Blackboard can be used not only to present materials that have been prepared and stored in advance, but also can be used "live" to put text and graphics on the screen. At any time, the instructor can move from a prepared page in the Electronic Blackboard to a "blank page" routine that permits the instructor to put upper and lower case text, numbers, and symbols anywhere on the screen by typing at the keyboard. Key terms can be underlined in color on the blank page by using keyboard commands to specify the color and the endpoints of the line. More complex graphics can be drawn "live" through the Electronic Blackboard's interface with the Apple Graphics Tablet.

All of the features of the Electronic Blackboard discussed above will be demonstrated during the paper presentation.

CONCLUSION

This paper has discussed and demonstrated the advantages of using an Apple II+ microcomputer and large-screen projector television as a replacement for the traditional chalkboard and overhead projector as the primary lecture aid for large classes. The advantages of this approach are its attractiveness to students, its legibility, and its use of color and animation to clarify complex materials and maintain student attentiveness. The Electronic Blackboard program, created to implement the concept, has been described and demonstrated, and experience so far with using the program to teach Principles of Economics has been reviewed. The Electronic Blackboard represents another step forward in utilizing the abilities of the computer, and the microcomputer in particular, to improve the quality of education.

ACKNOWLEDGEMENTS

My initial work on developing the Electronic Blackboard, and on modules for teaching Principles of Economics, was supported by NSF CAUSE grant #SER-80-04784. I would like to thank Dick Cornelius (project director), Mel Zandler, and the other faculty at Wichita State University who participated in the CAUSE grant, for help, encouragement, and good examples.

Apple, Apple II+, and Apple Graphics Tablet are trademarks of Apple, Inc. Image Printer is a trademark of C & C Software.

REFERENCES

1. Cornelius, Richard, "User Power; A Discussion and Real-Time Demonstration of Valuable Programming Features," NECC 1982 Proceedings, pp. 95-96.

2. Cornelius, Richard, "Microcomputers and Large-Screen Projectors in Science Lecture Halls," NECC 1982 Proceedings, p. 231.

3. Bowman, Barbara, and Randy Ellsworth, "Microcomputing in the College Classroom and the Effects on Student Attitudes Toward Computers," presented at the Annual Meeting of the Association of Psychological and Educational Research in Kansas, Emporia, Kansas, 1982.

4. Clark, James. E., "The Electronic Blackboard," presented at the Third Microcomputers in the Classroom Conference, Wichita, Kansas, 1982; presentation to IEEE Student Branch, Wichita State University, 1982.

5. Clark, James E., "Using Microcomputers and Large-Screen Projectors for Teaching Principles of Economics," presented at the Annual Meeting of the Joint Council on Economic Education, Kansas City, Missouri, 1982.

RESULTS AND LESSONS FROM A STUDY OF READERS' CONTROL OF RATE OF TEXT PRESENTATION ON COMPUTER SCREENS

Werner Feibel

Educational Technology Center
University of California
Irvine, Calif. 92717

ABSTRACT

In an effort to begin exploring the factors important for effectively using the capabilities of the computer to present text, two studies have been carried out. Results from the first one are reported. In this study, students in introductory college physics courses read three texts presented via computer screen. After each session, students were tested on the text content, and were interviewed regarding their reactions to the texts and the presentation on screens. During one of the sessions, each student was given control over the rate at which text appeared on the screen, including an option to stop the text at any point. While no significant differences were found between the control and no-control conditions, several tendencies were observed, and several useful lessons for research in this area were gained. These are discussed.

Introduction

As the computer comes to play an increasing role in schools and other learning situations, issues relating to the most effective pedagogical uses of a new tool and medium must be dealt with. To date, most of the research on these issues has concerned the use of the computer as a tool for testing learners' understanding of material presented, for providing them with additional opportunities to work with the material, -- e.g., in the form of drill and practice -- or for presenting tutorials on certain topics in the curriculum (although this use of the computer is still much less prevalent than its use for drill and practice).

Questions relating to the computer as a medium for presenting curricular material -- i.e., as an alternative to media such as books or films, particularly the former -- appear to be getting much less attention. To some extent this may be due to an implicit assumption that the resolution of most of these questions will take the same form it did in books. Several considerations, however, suggest that this may not be appropriate; moreover, it is by no means clear that the conventions and tendencies common for the printed page are the most effective ones possible, even for that medium.

There is a growing body of research literature on variables to be considered in formatting text on the pages of a book; these variables range from comparisons of different type styles to the most effective ways of laying out material on the page -- e.g., whether to justify margins, and how to use indentations and spacing to provide the reader with additional information and help for most effectively identifying major concepts and points (see, e.g., Hartley, 1980). While this research has not provided any clear-cut results to date, it has served to call into question a number of conventions and assumptions, and also to emphasize the complexity of the issues involved.

If we expect to get on the right track quickly with the new, electronic medium, such factors must be considered in the context of presenting text and pictorial information with computers. Furthermore, additional considerations enter the picture -- many of which have been discussed by Alfred Bork in his "Textual Taxonomy" (1981). In particular, the consequences of certain differences between the computer and the book must be taken into account. Two types of differences, temporal and spatial will be considered here.

In contrast to the information on a book page, which is static and completely present when the book leaves the printer's, the computer screen can be filled at a rate that the reader can select. This would give a reader the option of interacting with the material in a manner very different from a book. In addition to such peripheral possibilities as using this option to exercise and perhaps increase her reading rate, controlling the rate of information presentation could enable the reader to view material in a much more active manner -- e.g., by stopping the text presentation and trying to anticipate where a certain argument is leading, then testing her expectations (and thus, to some degree, her understanding of the material) by letting the subsequent text appear on the screen. Related to this is the possibility of animating diagrams and illustrations; the student could formulate hypotheses about a given situation and could actually view a simplified presentation of experimental results relating to the situation and discussed in the reading.

Besides these temporal factors, a second set of spatially-based differences between book page and computer screen as media concerns the cost, and therefore the use, of empty space on the presentation field. Blank space on a book page costs money, and is therefore generally avoided in this medium. One consequence of this is the relatively dense packing of print on the page. While economically advantageous, this may prove to be pedagogically detrimental. Blank space on computer screens, however, is free; this opens up many possibilities for using space on the presentation field to provide supplementary information about the relative importance of different parts of the material, as well as facilitating the formation of a visual image of the probable content interrelationships because one can easily scan a body of information repeatedly.

Thus, the luxury of free space can make various presentation possibilities more attractive. Dividing text into natural phrasing, thereby providing what is likely to be a more natural pace and chunk size for the intake of information, is one way of using space to possible pedagogical advantage. Formatting text in a way that makes clearer the hierarchical relationships in the material also opens up a large set of possibilities. While the latter, hierarchical, formatting depends directly on the actual content for its effectiveness, natural phrasing simply attempts to capitalize on processes and tendencies implicit in various informal learning contexts, but conspicuously absent in our educational system beyond the first few grades. I refer here to the use of meter for passing on oral literature and information, the timing that makes some speakers much more memorable and effective than others, and the use of motoric activity and rhythm to teach various skills and concepts -- (see, e.g., the work of Furth & Wachs, 1976, and that of Greenfield & Childs, 1976). We explored the possibilities of using such a resource to fuller advantage in learning settings.

As one step toward helping to clarify these issues and making decisions based on as much information as possible, we are carrying out two studies, in a project funded by the National Science Foundation, to investigate some of the variables relevant to the use of the computer as a medium for presenting textual material. The first study investigated the effects of giving students control over the rate of text presentation on their comprehension of the material and on their attitudes toward the learning situation. In the second study we investigated the consequences of natural phrasing on these same variables. I will present the results of the first of these studies -- discussing both the data and some of the lessons we have learned from this initial investigation. Some of these results and lessons have methodological implications for research in this area, and I hope they will be of use in guiding others in their own studies.

Procedure

Materials:

Portions of two chapters from a widely used introductory physics textbook were used in the studies. The goal was to find texts that were relatively self-contained, that would be of approximately equivalent difficulty, and that were considered equally interesting by readers. The actual texts were selected after pretesting them on students in physics classes and selecting the two that students regarded as most comparable in difficulty and level of interest. Certain, very minor, modifications were necessary to make the two readings sufficiently self-contained and stylistically acceptable; however, these changes were limited to the insertion of a few clauses, and the deletion of some short discussions of material related to other sections of the text. One reading dealt with electromagnetic induction, and the other covered the kinetic theory of gases. In addition, a third text from a different textbook, on the law of gravitation, was used in a practice session.

Tests designed to assess students' comprehension of various aspects of the material were constructed -- for the practice text and for the two readings used in the study. Questions ranged from those designed to determine the reader's recall of very specific information -- e.g., whether a particular phrase was used as a section heading, or where on the screen something appeared -- through understanding of various concepts -- e.g., defining the basic ideas and terms presented in the reading -- to grasp of the conceptual consequences of the material -- as reflected in students' ability to solve problems about the concepts and to discuss the significance of particular experiments for the topic under discussion.

Finally, a set of questionnaires was developed to obtain information about students' background with physics and computers, their attitudes toward computers, their study strategies and habits, and their reactions to being able to control rate of text presentation.

Equipment and Programs:

The texts were presented on a Terak 8510A personal computer. To allow the reader to adjust the rate of text presentation, Michael Potter, one of our student coders built several speed control boxes, which could be connected to the computer. The boxes operated by moving a knob to a position corresponding to the desired rate of text presentation; in addition, a button could be pressed to stop the text at any point -- until the reader restarted it by pressing the button again.

These boxes worked through a program written by Adam Peneschan, another student coder. The program translated settings on the box into timing delays or interrupts in the main program; these controlled the presentation of the text on the screen. Besides presenting the text on the screen, and drawing the accompanying diagrams, the main program also allowed the user to flip around to earlier or later sections in the text. This flipping could be done easily, and

students reported having little difficulty learning how to move around in the text. Finally, a program that recorded any changes in presentation rate and wrote this information to a data file for later analysis, became available during the study, and this was used to record the actual changes for some of the participants.

Participants:

Students were recruited from three introductory physics courses, and were paid for participating in three sessions. Most of the students were majoring in some field of science, although this was not a criterion for selection. Of the 33 students (23 males, 10 females) who participated beyond the practice session, 31 completed the study. Only two males dropped out.

Method

Students were tested over a six week period in a repeated measures design, with three sessions per student. When they arrived for their session, students were placed in a room with the computer, and the program was started for them. They were told to read the material, spending as much time as they wished on it. They were told that they would be tested on the material at the end of the session, and were permitted to take notes on the material. When they completed the reading, they were asked to give up their notes, and were given a test on the material. After this was completed, they were interviewed briefly about their reactions to the reading -- both as content and with respect to its presentation via computer.

The first, practice, session was not included in any comprehension analyses. Rather, this session was used to obtain background information on the students, and to give them practice reading text from a computer screen -- including developing familiarity with the means for flipping around in the text. Furthermore, the practice test administered at the end of this session informed students about the range of detail they were expected to attend to in subsequent readings.

After the first session, students were randomly assigned to either a learner control or no learner control condition for the next saession. Because of equipment difficulties, the division for the second session was 18 no learner control and 15 learner control. For the final session, students were put in whichever of the two conditions they had not done in their previous session. Because of attrition, the third session consisted of 14 no control and 17 learner control participants. Order of texts was fixed; all students read about electromagnetic induction in the second and about the kinetic theory of gases in the third sessions. Thus, roughly half the people had control over rate of presentation of the electromagnetic induction text, and the rest over the kinetic theory of gases text.

In the learner control condition, the student could adjust the rate of text presentation, or stop the text, at any point. Unfortunately, problems with the control boxes made the hold button somewhat unreliable, and most students reported that generally they did not use it beyond an occasional attempt -- generally to simply try it out. The control boxes permitted the reader to change the rate from a very slow, letter by letter, presentation of the text at one extreme, to a rate comparable to a reading speed slightly faster than the average reading rate for non-technical material. While this rate was regarded as sufficient, the students' reactions to this provided one of the lessons to be considered in research of this nature.

In the no learner control condition, the text was simply thrown onto the screen, so that the entire screen appeared in a couple of seconds. This rate was much too rapid for anyone to read it as it was coming out.

Results

While test scores in a sufficiently large class can generally be approximated quite well by a normal distribution, such an assumption was considered too risky in the present study -- because students actually came from three different classes, so that the underlying distribution would more probably consist of a combination of normal distributions. Consequently, analyses were done using nonparametric tests -- particularly normal deviates tests (see, e.g., Marascuilo & McSweeney, 1977).

On the whole, the major analyses identified no significant differences on the comprehension tests as a function of learner control. While differences in performance were generally in the expected direction, these differences were not sufficiently large to permit rejection of the null hypothesis. Thus, mean proportion correct on the text about electromagnetic induction was 0.565 (s.d. = 0.163) for the learner control students, and 0.529 (s.d. = 0.188) for the no learner control students; similarly, for the reading on the kinetic theory of gases, the figures were 0.654 (s.d. = 0.180) and 0.593 (s.d. = 0.150) for the learner control and no learner control students, respectively. Since Winsorization -- truncating the range of scores by discarding a certain number of scores at each tail of the sample distribution, in order to get rid of outliers -- made no substantial difference, the full set of available scores was used in analyses, to take advantage of the additional degrees of freedom available when working with the full sample.

The fact that the learner control individuals did score higher on both texts is encouraging, and speaks in favor of investigating this variable further -- particularly since the same students read both texts. This latter point makes it unlikely that the mean differences are attributable to the individuals or to the texts, since the relative standing of the two groups switched for the texts, and since the common denominator in the better performances was the learner control. Nonetheless, since the results only approached significance, they must be considered merely suggestive.

In the learner control conditions, students who checked more positive adjectives to describe their experience with computers tended to score slightly higher [illegible] students less positively inclined toward computers. The difference for the gas[illegible] was only about 1%, but again in favor of students with positive experiences with computers. The samples in the learner control conditions were too small to make it feasible to use this attitude information as a covariate in analyses. Nevertheless, these results make intuitive sense, and it will be helpful to explore the more general questions of the conditions under which computers facilitate learning or understaning in people with different experiences with computers.

Physics background did make a difference, as is to be expected, with those in the more advanced introductory course and those who had taken more physics courses performing better on the average; however, these differences again failed to reach significance. Moreover, this factor did not play a role in the grouping, since there were no differences in the make-up of the two groups (i.e., control with first reading versus control with second reading). The greater heterogeneity introduced through this broader range of physics background may have contributed to the overall results, however, since the increased range of performance attributable to these differences in familiarity with the subject matter would be expected to increase the variance of the performance figures.

Post hoc examination of subsets of comprehension test items (i.e., specific memory, definitions, concept memory, and problem solving) did not permit an unequivocal identification of the particular aspects responsible for observed differences. In part this was due to the much larger proportions of ties in scores on these item subsets -- because of the much smaller possible range of values. There did not appear to be any consistency in the types of items where the two groups differed -- something that might possibly be examined more systematically in a design where interactions can be studied directly.

Two other data are relevant here, since they also provide hints for subsequent research. First, many of the students in the learner control conditions said they found the continuous appearance of text on the screen distracting; most said they simply set the box to maximum and then waited for the screen to fill before actually reading the text. Related to this is the fact that many students mentioned the novelty of dealing with text presented in such a sequential and temporally variable manner. This novelty was not always perceived favorably. A more general issue raised by such reactions is considered below.

A second datum of interest concerns the finding that several of the students who considered themselves slower than average readers -- most of these being students with English as a second language -- found the learner control box useful. Again, however, these findings provide only suggestions for future studies. They cannot be considered primary results of the present study -- both because the number of people who considered themselves slow readers was too small, and because English as a second language was not explicitly included as a factor in the present study.

Discussion

Overall the results of the first study were inconclusive regarding the effects of learner control of text presentation rate on subsequent recall and understanding of the material. Nonetheless, several indications and lessons did result from the study.

First, for our purposes, the finding that performance on the two texts was comparable is encouraging for us, since it supports our intention of using the same texts in the second study. The average performance scores do not prove that the texts are comparable, but they do make such a claim plausible.

Other issues arising from the first study will help us in the details of running the second study, as well as providing some guides for other researchers in the area. First, the trends reinforce the importance of avoiding samples that are too heterogeneous. Possible solutions for this difficulty include either ensuring that all participants are drawn from the same classes, or designing the study to permit separate analyses for different groups within the sample.

The indications of differential effects of learner control as a function of attitudes about computers suggests that such information be obtained and either built into a multiple factor design as a grouping variable, or that the study be planned so as to permit the use of such information as a covariate in the analyses.

A more important and more general issue arises from students' reactions to the consequences of controlling presentation rate. College students belong to a group whose reading and study habits are largely developed, and likely to be relatively entrenched. In particular, this is a group whose members have all been reading for over a decade -- almost exclusively (if not entirely) from a printed medium. Thus, these people have invested a great deal of effort and practice in mastering the process of obtaining information from the printed page. Most college students are relatively proficient at reading, and they are quite likely to consider this aspect of their education as more or less completed. Therefore, hindsight suggests it should not be surprising to find them somewhat disconcerted and negative when required to deal with a situation that demands adjustment in some of these habits and in the expectations they have built up about the "behavior" of text.

The individuals who responded more favorably to this new situation -- those who considered themselves slow readers (and thus presumably feel a certain dissatisfaction on this score) and those who have been reading English for a shorter time than native speakers -- are students whose habits can be regarded as still changing, or at least more

amenable to change. These are people more likely to be receptive to the possibility of modifying their reading habits or of taking advantage of situations that can facilitate the process of gaining information for t[illegible].

An important implication of this consideration -- if correct -- is that such factors as control over rate of text presentation might be more profitably studied in individuals whose habits are still in the process of being perfected -- e.g., younger students or learners for whom particular reading situations are still relatively novel. In a sense, this implication can be considered a special case of the more general issue of the diffusion of a new tool or context -- something generally much easier with people who have less of a stake in what was previously available.

Such habits should be less influential in the second study, however, since there even those who consider themselves sufficiently competent readers will be dealing with a process very similar to activities in which they commonly engage --grouping information in a manner that seems natural.

The present study -- despite its inconclusive results -- does indicate several questions and directions in need of research. Before we make hasty and ultimately untenable decisions about the manner in which computers should be used in learning situations, we must be certain we can make those decisions from a reasonable and sufficient knowledge base. While I have been unable to add much specific information to that base, I hope I have provided some insight into some of the priorities and difficulties that must be considered as we move toward the computer age.

Acknowledgments

This research was funded by the National Science [illegible]ndation, through their Research in Science Education progra[illegible] (RISE Grant # SED - 8112378)

Many thanks to Kristina Hooper, Cynthia Powell, and Ruth Von Plum for their invaluable suggestions, insights and efforts. Thanks also to Michael Potter and Adam Feneschan for their efforts in making the speed-control box and writing the programs for the study. Without the efforts of these people, the project would never have come to fruition.

Bibliography

Bork, A. Textual Taxnomy. Unpublished paper, Educational Technology Center, University of California at Irvine, 1981.

Furth, H.G. & Wachs, H. Thinking Goes to School: Piaget's Theory in Practice. New York & London : Oxford University Press, 1974.

Greenfield, P.M. & Childs, C.P. Weaving, color terms, and pattern representation: Cultural influences and cognitive development among the Zinacantecos of Southern Mexico. Paper presented at the First International Conference of the International Association for Cross-Cultural Psychology. Hong Kong, 1973.

Hartley, James (Ed.) The Psychology of Written Communication: Selected Readings, New York: Nichols Publishing Co., 1980.

Marascuilo, L.A. & McSweeney, M. Non-Parametric and Distribution-Free Methods for the Social Sciences. Monterey, Calif. : Brooks/Cole, 1977.

An Experimental Comparison of Discovery and Didactic Computerized Instructional Strategies in the Learning of Computer Programming *

Brian McLaughlin

sion of Computer Research and Technology
National Institutes of Health
Bethesda, Maryland 20205

Abstract

This study compared the effectiveness of instructional strategies for teaching computer programming. College students, pretested for Locus of Control and cognitive ability, were assigned to use short instructional computer programs characterized by either a "discovery" approach or an expository "programmed instruction" sequence. All instruction and testing was administered by the computer. In general, expository instruction led to better posttest recall of basic programming facts while discovery instruction resulted in better performance on extrapolation tasks and actual programming. Discovery instruction led to higher self-confidence about newly learned information and a greater willingness to continue instruction. Students with an Internal locus of control performed better under discovery instruction, while External locus of control students did better under expository instruction. This interaction was remarkably consistent across eleven cognitive and affective outcome measures. An instructional program based on this research has been implemented on an Apple microcomputer.

Rationale

The psychological theories of Piaget and Bruner suggest the value of exploratory discovery-oriented instructional experience. The theories of Skinner and Ausubel emphasize more systematically controlled expository instruction. Ambiguity and controversy have long surrounded the comparison of instructional methods based on these two schools of thought. A core epistemological rationale for the discovery approach is the Piagetian notion of a learner interacting with the environment, actively constructing knowledge through continuous application and reorganization of personal cognitive structures. In contrast, the notion more often associated with the didactic approach suggests a more passive, dutiful learner who incrementally copies or absorbs new knowledge from an outside source. These two approaches might be broadly characterized as differing in their relative emphasis on "learning by doing" versus "learning by being told."

Despite its far ranging implications for both educational theory and classroom practice, the discovery versus didactic controversy has been only minimally illuminated by decades of empirical research. Bruner[1] asserted his broad claims for a discovery learning approach in the spirit of an hypothesis. In 1961 he stated that the hypothesis "is still in need of testing. But it hypothesis of such important human implica.ion that we cannot afford not to test it...". Twenty years of research have demonstrated the difficulties in testing Bruner's "hypothesis". Yet many important issues raised by the discovery - didactic controversy remain unresolved. By examining and improving upon methodological difficulties encountered in previous discovery learning research, this study aims to demonstrate the utility of a rigorous experimental approach to this intractable area of educational research.

Most experimental research comparing discovery and expository instruction has used some variant of "example-rule" methodology. The learner is given examples and must infer or "discover" the underlying rule. The narrow focus of this artificial laboratory implementation of discovery learning has unreasonably constricted the operationalization of discovery processes. In this study, a computer program was used to generate a discovery learning environment which provided opportunities for "messing about"[7] and had sufficient openness to accommodate the fuller exercise of intellectual and cognitive capabilities implied in broader conceptions of discovery learning[11,14,6].

Another methodological weakness hampering previous studies has been the all-too-common focus on a single instructional outcome, typically an achievement posttest. Proponents of the discovery approach have suggested a tantalizing array of resulting educational benefits. In this study, the effectiveness of instruction was assessed on a variety of cognitive, affective and motivational outcomes.

A third methodological difficulty has been the need for more sophisticated hypothetical models of complex instructional situations. As Cronbach[3,4] has repeatedly advised, it is crucial to carefully describe the specific instructional elements being investigated, the type of material being presented, the characteristics of the individual learners, and the nature of the outcome variables. In this study, specific instructional strategies were implemented as specific algorithms and routines in an instructional computer program. Focusing on a single subject matter area (computer programming), an Aptitude by Treatment Interaction (ATI) design was used to assess the relative effectiveness of instruction on learners with differring individual characteristics.

Method

A minicomputer was used to teach introductory computer programming to 81 college undergraduates using either a discovery or didactic instructional strategy. Learners had no previous computer programming experience. The didactic strategy used an expository programmed learning sequence. The question-answer-feedback format of this approach is representative of many current instructional applications of computer assisted instruction (CAI). The discovery strategy used a computerized discovery learning environment incorporating instructional heuristics and epistemological assumptions of the discovery approach. The discovery environment consisted of short model programs which could be learner-modified and executed. During (deliberately slowed) program execution, the computer screen displayed simulated dynamic computer transactions. The amount of personal initiative required of discovery learners was varied by prohibiting or encouraging learners to initiate their own changes in the model programs. In all discovery treatments, the visible immediate feedback provided by executing programs and program variations provided the opportunities for learning.

Instructional treatments (each approximately one hour in duration) were compared for effectiveness on eleven outcome dimensions (derived from these: multiple choice posttest items, program generation tasks, confidence ratings, Continuing Motivation measures[9], risk-taking measures, and general rating scales.) Two individual difference variables were examined for possible Aptitude by Treatment Interactions: (1) Internal-External Locus of Control[13], ("IE"); and (2) a measure of cognitive ability ("Ability", based on the SAT Verbal score and an algebra word problem test.)

Data Analysis and Results

Data analysis was performed using stepwise multiple regression techniques as suggested by Cronbach and Snow[4]. There were four experimental treatments (N=20 for each treatment group):

1. Discovery High Initiative (learners made own changes)
2. Discovery Low Initiative (learners made only suggested changes)
3. Discovery Optional Initiative (learners made own and/or suggested changes)
4. Expository Programmed Learning (branching programmed instruction sequence)

Treatment effects were partitioned into the three orthogonal treatment contrasts of most theoretical interest. Independent variables were entered in the prediction equation in a predetermined order: Ability, IE Locus of Control, pretest covariates (derived from an identical training session given all subjects), treatment vectors, then second order interaction vectors, and finally third order interactions. Using this "step-up" regression strategy, the increase in R squared due to each successive term was tested for statistical significance with F ratios constructed using the error term of the final full model. The results are summarized in Table 1.

Aptitude by Treatment Interactions were interpreted using graphic representations and the Johnson-Neyman technique[8,12]. An example of such an interpretation is illustrated in Figure 1. Posttest Effort represents the subject's response to a final rating scale: "How seriously did you try to answer all the previous questions?". After being residualized on other significant predictors (in this case, a better (faster) training score predicted higher reported effort), Posttest Effort was separately regressed on IE and Ability. The resulting disordinal interaction obtained for IE is sketched in Panel 4 (bottom right quadrant) of Figure 1. Construction of a Johnson-Neyman ($p<.05$ level) region of significance indicated that for subjects with IE scores less than 8.9, discovery instruction was significantly more effective while for subjects with IE scores greater than 16.2, Programmed Learning was more effective.

Interpretation of significant three way interactions (Ability by IE by Treatment) was aided by three dimensional figures. For example, the results for Confidence in Right Answers (confidence ratings after each posttest item were averaged separately for items answered correctly and incorrectly) are illustrated in Figure 2. Comparison of corresponding corner points of the two surfaces in Figure 2 helps summarize the character of the interaction:

1. Discovery was more effective for Internal low Ability subjects and External high Ability subjects.
2. Expository instruction was more effective for Internal high Ability subjects.
3. The most striking difference between the two surfaces was the exceptionally poor performance of Internal low Ability subjects under expository instruction.

Summary and Conclusions

The differences among discovery treatments were more suggestive than dramatic. However, these within discovery comparisons did demonstrate the practicality of comparing variations in the discovery approach. Experimental control over instructional elements of discovery instruction offers numerous departures for further research and promises continuing refinement of a computerized discovery approach.

In contrast to within discovery comparisons, the differences between discovery and expository programmed learning were substantial. These results can be summarized in five general conclusions.

1. Expository instruction led to superior reception and recall of basic programming facts and rules. Discovery instruction led to better integration of newly learned material as evidenced by superior performance writing programs and on multiple-choice posttest items requiring interpretation or extrapolation. It is interesting to note that if all cognitive outcome items had been lumped together as a single achievement posttest, the differential performance on each subskill would have gone undetected. The failure to separate posttest items of different types may be one contributing reason to the dearth of

Table 1

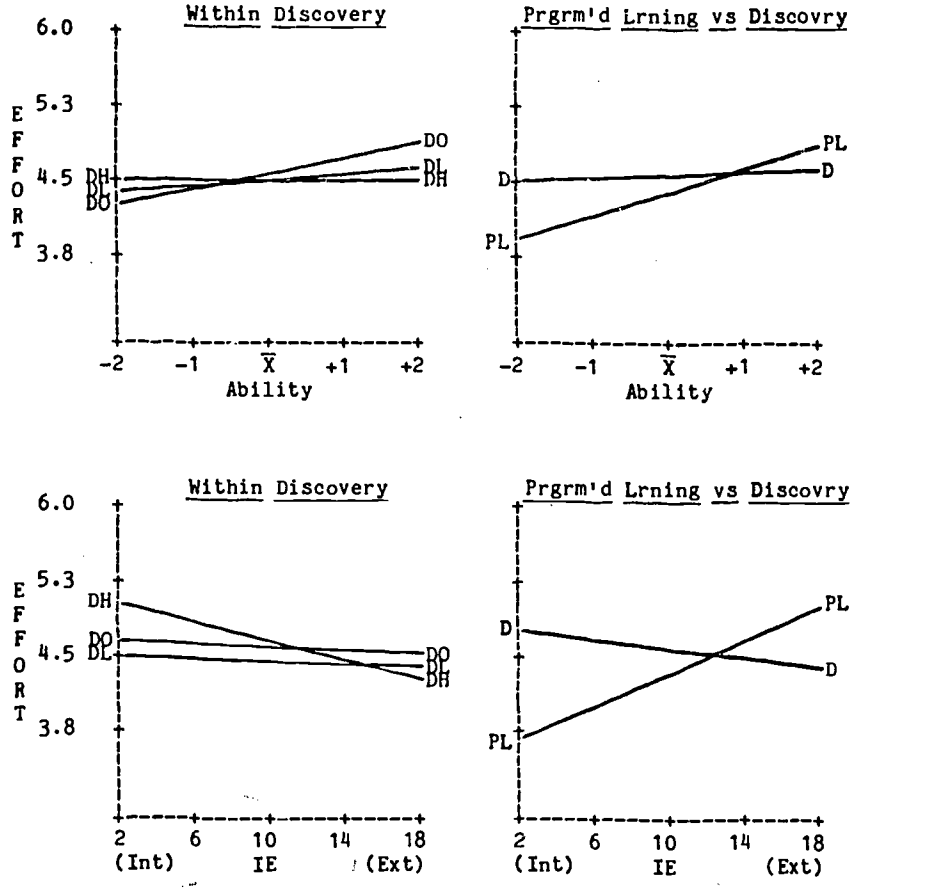
Increase in Percent of Outcome Variance Accounted for by Each Term
(Values Below 1.0% Not Displayed)

Outcome Measure	P	N	Full Model	A	I	P	T1	T2	T3	AT1	AT2	AT3	IT1	IT2	IT3	AI	AIT1	AIT2	AIT3
Near Transfer		81	30.6^{p}	16.7^{q}		XXX	1.7		4.1^{p}	3.1					1.9	XXX	XXX	XXX	XXX
Far Transfer		81	32.3^{q}	12.7^{q}	5.7^{p}	XXX	8.3^{q}	2.6								XXX	XXX	XXX	XXX
Generate Programs		81	32.1^{q}	25.7^{q}		XXX			4.5^{p}							XXX	XXX	XXX	XXX
Confidence on Wrong Ans	a	78	24.5	7.0^{p}		9.1^{q}						1.7			1.7	1.9		1.2	
Confidence on Right Ans	b	76	41.7^{q}	16.6^{q}		7.2^{q}			6.9^{q}			2.1							4.6^{p}
Confidence in Programs	b	76	34.0^{p}	11.8^{q}		3.4	2.0		9.5^{q}						2.8	XXX	XXX	XXX	XXX
Risk Taking		81	38.8^{q}	17.2^{q}		XXX					2.8	1.6			6.4^{p}		2.9		3.7
Posttest Effort	a	74	29.8^{p}	4.7^{p}		5.4^{p}			1.1			2.5	1.0		12.4^{q}	XXX	XXX	XXX	XXX
CM	c	78	32.4^{p}			14.2^{q}					4.6^{p}		1.4	4.2	2.[illegible]		2.5	2.3	
Request Information	c	78	25.6		2.1	2.9			4.6^{p}		9.8^{q}	1.7			3.5	XXX	XXX	XXX	XXX
Take Regular Course	b	76	31.1^{p}			5.6^{p}		1.6	4.8^{p}	2.3	3.9		1.7	3.6	4.8^{p}	XXX	XXX	XXX	XXX
Positive Affect		81	30.6^{p}	1.2		XXX	1.0	2.6			5.5^{p}	6.7^{p}			4.0	1.9	1.8		5.9^{p}
Instruction Time		60	34.1^{p}	15.5^{q}		XXX			5.2		1.3	5.6	3.2		2.2	XXX	XXX	XXX	XXX

Significance of F Test : p=p<.05 q=p<.01 XXX = Not in Model
A = Ability I = Internal-External Locus of Control (IE)
P = Predictor: a=Training Score b=Pace c=Key Count
T1 = High Initiatve Discovery versus Low Initiatve Discovery
T2 = Optional Initiative versus Mean of High and Low Initiative Discovery
T3 = Programmed Learning versus All Discovery

Figure 1

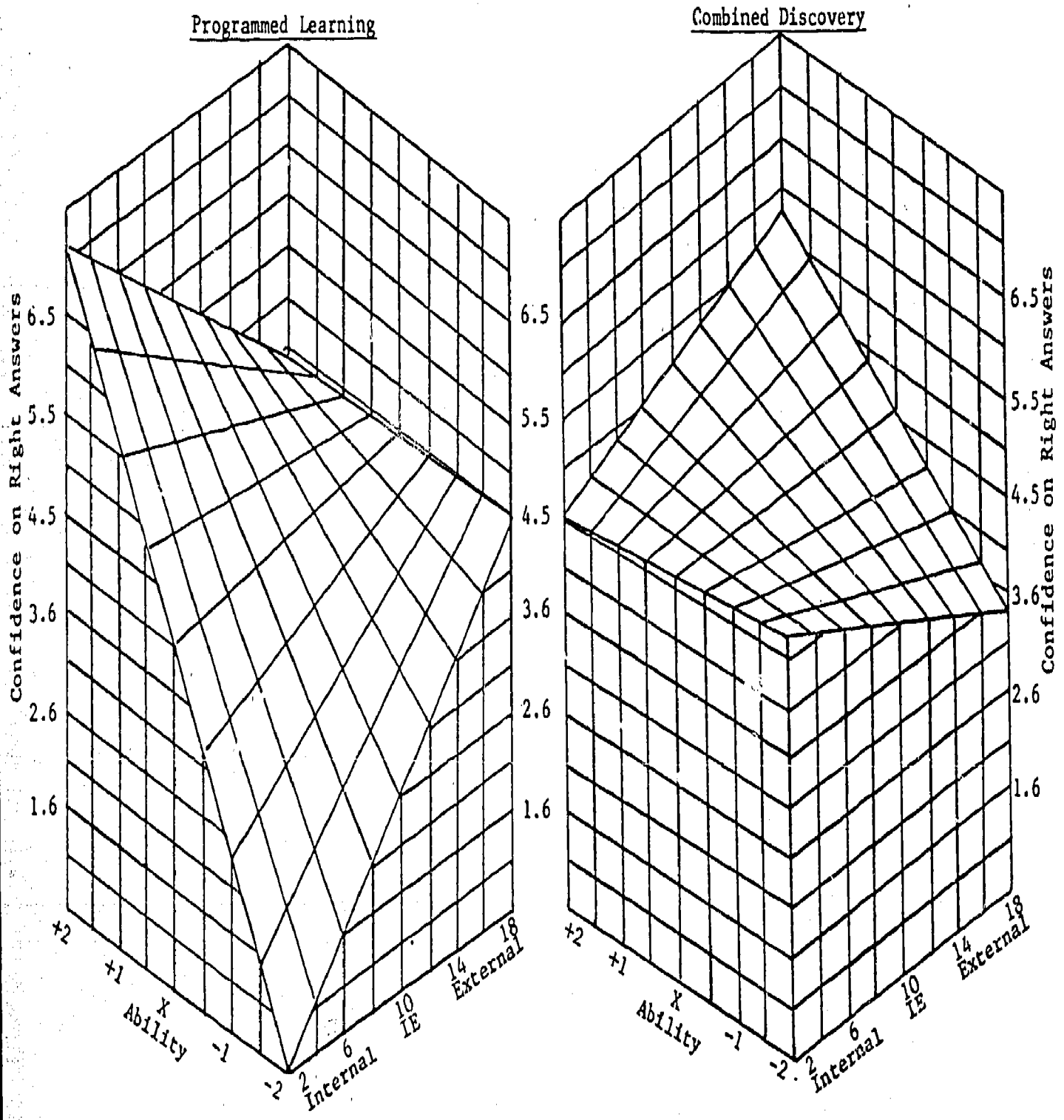
Separate Regressions of Ability and IE on Posttest Effort.



Posttest Effort : high="tried hard to answer all questions correctly"
low="did not care about answering questions correctly"

Each X and Y axis unit = 1 standard deviation
Int = Internal Locus of Control Ext = External Locus of Control
DH = Discovery High Initiative
DO = Discovery Optional Initiative
DL = Discovery Low Initiative
PL = Expository Programmed Learning
D = Combined Discovery (DH, DO & DL)

Figure 2 Regression of Ability and IE on Confidence on Right Answers



Rescaled regression equations: Programmed Learning $Y' = 4.18 + .85\text{Ability} + .19\text{IE} - .44\text{Ability} \times \text{IE}$
Combined Discovery $Y' = 4.63 + .27\text{Ability} - .01\text{IE} + .17\text{Ability} \times \text{IE}$

Note: each scale unit represents 0.5 standard deviations.

consistent findings in previous studies of discovery learning. The results of this study strongly support the work of Mayer[10] in emphasizing the differential structure of cognitive outcome.

2. Discovery instruction led to higher self-confidence ratings, but only on correctly answered posttest items. This is interpreted as indicating superior cognitive discrimination after discovery instruction. This critical ability to discriminate levels of certainty about new knowledge is seen as an important (and neglected) educational outcome.

3. Discovery instruction led to higher levels of expressed interest in continuing computer programming instruction in additional educational settings. This supports Maehr's[9] conceptualization of Continuing Motivation as generalizing beyond the liking of instructional method to increased interest in instructional content. This finding also supports the motivational implications of discovery learning as expressed by other researchers[1,5]. Discovery instruction appeared more capable of "turning-on" learners than did expository instruction.

4. The results on affective and motivational outcomes indicate that lower Ability learners benefited more from discovery instruction than from expository instruction. Additionally, in contrast to many previous findings, lower Ability learners showed the same level of cognitive achievement (relative to higher Ability learners) regardless of discovery or expository instruction. Thus this study suggests that discovery can be an effective and promising strategy for teaching computer programming to lower Ability learners.

5. The results for IE Locus of Control showed that Internals did better under discovery instruction while Externals did better under expository instruction. This interaction was particularly evident on motivational and affective measures but was remarkably consistent across all eleven outcomes. These findings confirm the relevance of Internal-External Locus of Control as an important variable in educational research.

In summary, this findings lend definite empirical support to important expectations of discovery learning theory. Despite the relatively short duration of instruction, a substantial number of significant treatment differences were observed. The use of computerized learning environments is a flexible and promising research approach for the controlled investigation of instructional processes.

An instructional program based on this research has been implemented on an Apple microcomputer. The program, named Sherman, provides self-instruction in the Pilot programming language using a discovery oriented approach. It requires an Apple IIe computer or an Apple II with 64K and an 80 column board.

References

1. Bruner, J.S. The act of discovery. Harvard Educational Review, 1961, 31, 21-32.

2. Cronbach, L.J. The logic of experiments on discovery. In L. Shulman & E. Keislar (Eds.) Learning by discovery: a critical appraisal. Chicago: Rand McNally, 1966.

3. Cronbach, L.J. Beyond the two disciplines of scientific psychology. The American Psychologist, 1975, 30, 116-127.

4. Cronbach, L.J. & Snow, R.E. Aptitudes and instructional methods. New York: Irvington, 1977.

5. DeCharms, R. Enhancing motivation: change in the classroom. New York: Irvington, 1976.

6. DiVincenzo, R.M. Forming a theoretical synthesis for viewing discovery learning instruction. School Science and Mathematics, 1980, 80, 218-226.

7. Hawkins, D. Messing about in science. Science and Children, 1965, 2, 2-6.

8. Kerlinger, F.N. & Pedhazur, E.J. Multiple regression in behavioral research. New York: Holt, Rinehart, & Winston, 1973.

9. Maehr, M.L. Continuing motivation: an analysis of a seldom considered educational outcome. Review of Educational Research, 1976, 46, 443-462.

10. Mayer, R.E. Different problem-solving competencies established in learning computer programming with and without meaningful models. Journal of Educational Psychology, 1975, 67, 725-734.

11. Papert, S. Mindstorms: children, computers, and powerful ideas. New York: Basic Books, 1980.

12. Rogosa, D. Comparing nonparallel regression lines. Psychological Bulletin, 1980, 88, 307-321.

13. Rotter, J.B. Generalized expectancies for internal versus external control of reinforcement and decision time. Psychological Monographs, 1966, 80, (No 1, Whole No. 609).

14. Strike, K.A. The logic of learning by discovery. Review of Educational Research, 1975, 45, 461-483.

* Note: this article is a summary of the author's doctoral research at The Catholic University of America. The dissertation based on this research was among five finalists for the American Educational Research Association's Outstanding Dissertation Award for 1982. The instructional computer programs used in this study were originally developed by the author to run on a PDP-11/34 minicomputer.

CHECKING LAB CALCULATIONS

William F. Pelham

Physics Department, Towson State University,
Baltimore, Maryland 21204

Abstract

Computer programs were written, as APL functions, that checked every single calculation made as part of the required laboratory work in a two semester General Physics course. Both the values and the roundings of the results of the calculations were examined. Students reported individually, in an interactive mode, and received immediate evaluations; records were kept of their entries and successes. A description of the work, the bases of the programming, and some results of a six semester trial are given.

Students attending a required laboratory session in General Physics courses are presumed to know the purpose of the work to be performed, to have read the directions, and to understand the theory behind any calculations to be made. Typically, some instructions are given them before starting work, particularly on the use of equipment, and the experiments are then conducted. After the work is completed, a written statement, following the raw data, giving the "results" of the experimental work is prepared in the laboratory notebook. The notebooks are periodically submitted for evaluation, and the contribution to the course grade of the laboratory work is largely based on these evaluations.

Almost all the laboratory exercises involve using the data in calculations; these vary in complexity, but are usually quite simple in form. It is in doing the physics, relating the data to the physical concepts being used, that students have trouble. Using the wrong force, or the wrong current, for example, will give numerically wrong results that may not look suspiciously different from the right ones.

When the notebooks are evaluated, often by student assistants, every calculation is not repeated and there is, thus, the possibility that erroneous results, deriving from faulty physics, will remain undetected. The power of the laboratory to reenforce and explicate the lecture can be, clearly, compromised. A way to minimize undetected calculation errors is to have a computer perform the same calculations, using the student's own data, and compare results. This gives an immediate evaluation and has the advantage of doing it while the work is still fresh in mind. A commitment was made to develop systematic procedures and computer programs that would check lab calculations. The remainder of this paper will describe the general plan for doing this, pedagogical decisions made, several results of a six semester trial, and some conclusions.

General Plan

Every single calculation performed by every single student would be checked for numerical correctness and proper expression. Proper expression means the correct number of significant digits in the result of a computation as determined by the rules of significant digits and the uncertainties of the contributors to the computation.

In an interactive mode, then, a student would enter raw data, carrying only one uncertain digit, and the results of the requested computations. The computer would perform the same computations and compare its results with those of the student. Questions requiring discursive answers could be asked at appropriate points. When all questions were answered, or the student elected to stop, the computer would prepare a summary of the student's success and it, along with everything the student entered, would be stored. The instructor may examine the stored material for grading purposes and for helping students see where they went wrong.

Pedagogical Decisions

Before programming could start, certain questions needed answers. The list that follows gives most of the questions and the answers used in guiding the construction of the programs:

1. When in the computing session, if ever, should a student be told whether his and the computer's results agree? Immediately? At the conclusion of the session?

   Answer: Immediately; first, whether the numerical result is correct and then whether there is a significant digit error.

2. Should a student be given a chance to correct an erroneous result, and, if so, how many times?

   Answer: Yes, unlimited times.

3. Should the student be given the computer's results?

Answer: No for the numerical part, yes for the rounding, but only on request and with a concomitant loss of credit.

4. May a student redo all or part of the report during the current session or a later one?

Answer: Either way. A count will be kept of the number of recordings of the report.

5. May a student stop before a report is completed and finish it at a later date?

Answer: Yes.

6. Should there be a penalty for late reports?

Answer: Yes.

7. Should the computer attempt to provide instruction?

Answer: No.

Programming

The entire operation was to be run on the available UNIVAC 1100/10 computer through dial-up access with DECWRITER II terminals; the terminals were not, and could not be, located in the laboratory. APL was to be used to write all the programs (henceforth called APL functions) because it seemed especially convenient for the task being considered. Doubtlessly, the special features of APL affected the programming strategies used, but that is probably the case whatever the programming language.

Some features of APL and how they were employed will now be given:

a) The APL environment itself, with its ability to have numerous functions, sitting in an easily retrievable workspace, that can be called as utility functions by a master function. All the digit counting, rounding and file usage were handled this way.

b) APL random access files into and from which multi-dimensional arrays can be written and read as a single variable. Each student's report was stored and retrieved, with one instruction, as a 3 dimensional array containing 3-2 dimensional matrices of, typically, 26 rows and 6 columns each.

c) The ease of locating and selecting data elements from both character and numeric vectors and matrices. Each student entry was a character vector and it was examined for errors, and the digits of each number counted before conversion, via the execute operator, to numeric data. The summary of the student's success was also simplified by these features of APL.

d) The ability of APL to perform parallel processing facilitates the entering and handling of repetitious data. When 5 runs were made, all 5 calculations were done simultaneously with one instruction, greatly shortening the function. Indeed, the compactness of APL results in short functions, a kind of dense pack of instructions, albeit sometimes hard to read in detail, but relatively easy to follow as far as the flow of thought is concerned.

As hinted above, each student entry was examined for the number of significant digits and the place value of the uncertain digit. Up to 5 items (an arbitrary decision) were entered on one line, so entries were stored, temporarily and in a file, as 6 column matrices. Each calculation got its own row. One matrix was used to store data and results, one to store numbers of digits, and one for place values; these were catenated into a 3 dimensional array before storage in an APL file.

Security was maintained by having the functions that read from and wrote into the file locked. Locked APL functions cannot be listed by anyone including the locker. The file name, consequently, remained unknown to the students. To know what was in the locked functions meant that unlocked copies had to exist somewhere out of students reach, and they did. To get at them required a knowledge of 4 different keys and the instructor's personal ID. If one was not experienced in APL, it also meant the ability to decipher the UNIVAC APL Users Guide.

The master functions, one for each lab exercise, were stored as elements in a UNIVAC program file and written with digraphs rather than the special APL symbols. This was done originally to avoid filling an APL workspace, but turned out to be advantageous in that the UNIVAC Editor could be used to correct and update functions, a much more efficient method than using the available APL function editing procedures. When it came time for a particular master function to be used, it would be added to the runstream, inside the APL processor, as a file element with the UNIVAC ADD command. Since an APL command was also in the file element, the reporting process was initiated automatically upon completion of the ADD.

A sample of a student report together with a few annotated lines of programming will be found in the APPENDIX.

Results of Trials

As the reporting system began to be used by the first of the total of 350 students, unsuspected problems arose, some of which could be corrected and some of which await solution. Perhaps the most frustrating occurrence for students was to make a typing error of a sort that would cause the execution of the function to be suspended; this happened often in the beginning. In APL, if a statement cannot be executed, an error message is printed and the number and content of the unexecutable line follow. The system then waits. Things are not stopped, however, merely suspended, and the function can be restarted at any line with an appropriate command. All variables remain intact. Attempts to instruct the students how to restart the function were not completely successful. (They were to type $GGOBACK whereupon the response would be "To Where?" Their reply was supposed to be the

question number to which they wished to return.) The main problem seemed to be a lack of recognition that the function had been suspended. None of the students was familiar with APL. A special error detection function was written to help this problem and it has somewhat. Other solutions will be attempted.

Another unexpected problem was that many students refused to believe the computer. They were right and the computer was wrong. They would repeat the same entries over and over sometimes with slight changes. They would claim that their lab partners had gotten the exact sequence of entries declared correct that, for them, were declared wrong. After hours and reams of paper, they would finally slap the print-out down on the instructor's desk and essentially dare him to prove them wrong. He virtually always would and it was almost always because the students had made physics or digit mistakes.

The above circumstances inevitably arose with the less able students. The good students, who had everything right before they started, breezed through the reports. The unsure students would start skipping about, calculator in hand, changing earlier entries until, after a while, they lost track of what data the computer was using to make its calculations and consequently accused it of being in error. By the end of the second semester of the course, the general level of the discipline necessary to track through the print-outs had improved, but the full acceptance of the need to be completely logical about a sequence of steps was not universal.

Another unwholesome situation developed when % differences were being calculated; students were instructed to express the results of such calculations to only one digit (the fact that they would round 0.62% to 1% and throw up their hands at 13% didn't help matters). When the computer would tell them for, say, the fourth time that their 5% was wrong, they would resort to trial and error: first 4, then 6, then 3, then 7, and so on. Something will have to be done about this.

Good things happened, too. Misunderstandings were brought to light, even with better students, that the reading of a lab notebook would not have detected. There was a drive to get it right, instilled by interacting with the computer, that a red X in a notebook would not create. Consultation with the instructor to see why the computer called them wrong, and the subsequent acceptance of the correct result provided considerable satisfaction to many students.

Upon informal inquiry, it seemed that the better students liked reporting their results on the computer. For students poor in physics and/or for those few who seemed permanently bewildered by the process of using the computer, computerized lab reporting cost them hours of time; a reduction of this waste is needed and will be attempted.

Conclusions

Some of the answers to the pedagogical questions given in an earlier portion of the paper may have to be changed. Unlimited ability to make changes, in particular, seems to be in need of revision. This is what ultimately uses the students' time; perhaps an automatic review of entries or a referral to the instructor after a few tries is what is needed.

Writing the functions and continuously improving them is sometimes tedious, but is not unpleasant and is recommended. A rethinking of the entire scheme seems now due, particularly with a view to making it easier to add and subtract questions; if other faculty members, not adept in APL, are to use the reporting system, it is essential that tinkering with the questions be as easy as possible.

Merely checking the results of calculations is, obviously, not the whole of evaluating a student's learning in required laboratory exercises. Other aspects of laboratory work (preparedness, quality of work, follow-up) can be part of a computer managed laboratory. The insistent nature of the computer's responses and the motivation to succeed that it induces suggest that learning will be improved through computerization of laboratory work. To have a "go-no go" test of preparedness to perform a lab would, however, require on-line computation in the laboratory and, most likely, a so-called "open" laboratory schedule; i.e., a schedule that permits students to work in the laboratory at virtually any time. The work being reported can easily be adapted to the open lab concept and was undertaken with that possibility in mind.

Appendix

First, some printout of a typical student interaction. Q1 is straight data input, Q2 is the result of a calculation, correct, in this case, Q6 shows some student errors and Q7 shows the reintroduction of previously entered information.

```
>@ADD CML.REPORT20306
apl/1100 level  7r1bv2 tue 03/01/83 16:20:39
clear ws
good day garrett , if you are all
set to report a lab, type ok, if not, type stop.
>OK
it is 2 days after the lab was performed, and you
are:
on time in reporting.
this is to report lab 6
type done to stop, delta to change place.
type y to have sigdigs evaluated, n not to (with
lower grade).
>Y
Q1:first, we'll check the calculation of the
friction
force. enter fup, fdown (in newtons) and theta, in
radians.
>1.62 .539 .524
Q2:give the friction force in newtons.
>.541
```

```
calculations all ok
sig digs all ok
Q6: give the change in potential energy, in j, as
the block moves between the two beams, run 1, run
2.
>.2320. .2020
incorrect values are 2nd
do you want to reenter it (them) (y or n)?
>Y
>.222
there is a sig dig error in the 2nd entry(ies)
do you want to replace it(them) y or n?
>N
do you want to know the correct form(s), y or n?
>Y
2nd entry should have 4 digits.
Q7: and now the ke change + the friction work, in
j, run 1, run2.
>OLD
print old entry?  y or n.
>Y
 2.20e&1 2.20e&1
 calculations all ok
sig digs all ok
last chance to make a change. type done or delta.
>DONE
done--your work has been recorded
you will now be automatically signed off apl.
sign off system or @add report another lab.
apl terminated
```

Next are the master function lines associated with Q7.

```
80:Q7:'Q7: AND NOW THE KE CHANGE + THE FRICTION
WORK, IN J, RUN 1, RUN2.
81:PL$S'7'
82:NUM$S2
83:ENTER
84:$QACT
85:ANS1$SKF$SENT
86:ANS$S-/PQ$S.5$XSLM[3]$X(SLM[2]%T[2 1])*2
87:ANS$SANS,-/WR$S.5$XSLM[3]$X(SLM[2]%T[4 3])*2
88:ANS$S ANS+FF$XSTAR
89:N$S$C/PQ RNDNP $L/MT[4;1 2],MT[3;2]
90:N$SN,$C/WR RNDNP $L/MT[4;3 4],MT[3;2]
91:WW$S(FF$XSTAR) RNDNP MT[2;1]$LMT[5;1 2]
92:ANS$SANS ROUNDP N$SWW$CN
93:ITEM$S8
94:RETEST
95:N SIGCHECK 'AA'
96:KF$SANS1
```

Bear in mind that APL goes from right to left. Line 85 is the student's entry called KF and ANS1. Lines 86, 87 and 88 calculate the "correct" result from data entered earlier; it is a 2 element vector, unrounded, called ANS.

Lines 89, 90 and 91 compute rounding information and line 92 rounds ANS correctly. The function RNDNP takes the quantity to the left of its name and rounds it to the number of digits to the right of its name ($L/ selects the smallest number of digits from the group of 3 to its right); the function returns the place value of the rightmost digit of the rounded quantity. ROUNDP rounds the left quantity to the place value given on the right and returns the rounded number. Lines 86 through 92 are doing the same calculations on two numbers simultaneously.

TEACHING UNDERGRADUATES TO THEORIZE THROUGH THE USE OF A COMPUTER SIMULATION OF KIDNEY FUNCTION.*

by David L. Wilcox

Biology Department, Eastern College
St. Davids, Pennsylvania

Abstract

Using a simulation of osmoregulation, ninety seven physiology students in four different classes engaged in student research projects, deducing model structure through open ended experimentation. They each developed theories, designed experiments, interpreted results, and reported to the class in journal format. Each class made considerable progress in deducing model structure, developing such typical traits of the scientist as doing extra (no credit) experiments and partisanship of certain theories. They debated their opinions concerning kidney control mechanisms with great enthusiasm, and they learned a considerable amount about kidney function.

Introduction: Creativity and Simulation

Often we are victims of our own success in the teaching of science. Because the current "state of the art" in science changes so rapidly, we spend most of our time explaining current formulations, and our students know little of the thrill of discovery. However, a number of writers[1] have demonstrated that the quality of independent student investigation such as publications, senior theses, or even science fair projects, are distinctly better predictors of students' future productivity in their discipline, than such traditional measures of student ability as the SAT, GRE, or college grades.

Directed student research is, of course, costly both in institutional funds and in instructional time, and it may reduce the content which students will master. It seems likely, however, that neglecting of the needs of our more creative students may be even more costly, and may reduce the number of really creative students who choose to enter or remain in science[2].

*Under partial support of N.S.F. LOCI grant: SER 79-00115.

A partial solution to this problem may be the use of simulations in investigative labs. Simulations provide a powerful substitute for certain investigations for which one does not have the time, equipment, expertise or moral right. Due to the speed of the process, and to the limited number of variables which can interfere with the experimental process, the computer simulation of biological systems may be used to give the student experience in the process of scientific discovery within the time constrants of a single course. Such open-ended experimentation introduces students to scientific thought at a level not usually experienced before graduate school. In a time of limited educational funds, simulation may be the only affordable way to allow students to repeat an experiment and learn from their errors[3].

Simulations may be used as laboratory exersizes in various ways, ranging from simple demonstration to the design of new mathematical models. A few notable examples include: the demonstration of system controls, the deduction of system relationships, an effort to control the system, the deduction of parameter values, the identification of unknowns, the upgrading of present models, and the design of new models.

The Design of the Kidney Simulation

The project being reported in this paper used a simulation of mammalian fluid balance as a system for open ended laboratories. The program was originally designed to run on an Alpha-Micro mini-computer with time sharing capacity, using five CRT terminals and a TI 810 printer. Each CRT is assigned 32,000 bytes of dynamic memory.

In writing the "Kidney Project" simulation, I used the MENTOR simulation system, designed at Eastern College. (A short version of MENTOR has recently been written for the Apple II.) MENTOR consists of a modular base upon which models using sets of difference equations may be simulated. Besides its availability to us, its most important advantage was its very high

flexibility in experimental design, necessary if students are to effectively form and test their own hypotheses. In MENTOR simulations, the user (student) sets the length of the experiment and the time interval between observations, chooses the observations, changes the number and the size of the experimental groups, sets times at which values may be changed, and assigns the values for any and/or all of the simulated system's parameters. This flexibility allows an almost infinite number of possible experiments. In addition, the MENTOR system supports its simulations with a package of 'utility' programs to analyse and graph data, advise on experimental design, and store or copy data files. Additionally, the interactive nature of the system enables a student with ten minutes of introduction to work on a simulation without supervision.

The computer program, "KIDNEY", simulates a terrestrial mammal's homeostatic controls of fluid and salt levels. As the compartment model in Fig. 1 shows, the model has four interacting "flows" of material: the two substances controlled, fluid volumes and amounts of sodium; and the two controlling hormones, anti-diuretic hormone and aldosterone. I assumed that the water reabsorbed from the nephron comes from two locations, the convoluted tubules (including the aldosterone sensitive site in the distal convoluted tubule) and the collecting duct. Sodium reabsorption I assumed to be a function only of the first location.

Fig. 2 shows the major variables of the model linked in a signal flow diagram. Flows and levels of sodium are given in percentage (meq.). Cardiovascular conditions control hormone synthesis, and the hormones in turn control reabsorption. Glomerular filtration was a function of blood volume and several other cardiovascular parameters.

The model has 41 parameters, and can be reduced to eight differential equations (Appendix 1) representing the blood levels, the amounts in the intake and urine collection "jars", and the hormonal levels. Observable variables given students include the sodium concentrations and fluid volumes of the intake, the blood, the glomerular filtration, the flow in the convoluted tubules, site specific reabsorption and the urine. The model also includes diurnal changes in GFR, and tolerance limits for blood volume.

KIDNEY has been used to allow my physiology classes to pursue a semester long class research project. Each class had the goal of deducing the structure and the control loops of the kidney model. Starting from scratch, every conclusion had to be supported directly from their data. At monthly intervals, each student independently designed, ran, and "published" an experiment in an in-house journal, "The Kidney Project Record". Each paper thus became primary literature for subsequent student research. Students could also write "letters to the editor" for extra credit. Following each set of papers, the class discussed their current understanding of the model.

Results of the Use of the Kidney Program

Variation in and between classes usually makes the evaluation of an educational innovation very difficult. However, this project produced products which could be analysed and evaluated as concrete entities, student scientific papers. The results clearly demonstrate the scientific process going on in the class.

Fig. 3 shows three figures from a student's papers (Brauch, 1979), summarizing the current thinking of the class at 3 points during the semester. Her diagrams clearly show the development of theory in the group of students. There was an average of eight new relationships "discovered" with each new set of papers. Although total knowledge increased, it was not by a smooth accumulation of data. Even in this simple system, students sometimes disagreed in their interpretations of data, and there were some "contradictory" results. Even "test" experiments might not resolve debates, and new parameters and complexities were discovered.

A good illustration is the changing view of the relationship between intake rate and the blood volume. The student researchers initially thought this a simple, direct relationship (Fig. 3a). By the second paper this consensus had broken down, and for most of the rest of the semester my class had two "schools of thought" about the effect of fluid intake rate on blood volume (Figs. 3b and 3c). The confusion came from neglect of the indirect effects of salt intake vs. water intake. Instead of immediately correcting them, I let them argue. Fig. 4, which shows the final understanding of several of the students, demonstrates the results. Eventually, most of the class was convinced by the "negative feedback" school (which was correct). The speed of simulated studies allowed the experimental process to continue long enough to be self correcting. This pattern of changing thought is remarkably similar to the ideas of Thomas Kuhn on the process of science[4].

By the end of the study when the illustration in Fig. 4 were prepared, most of the students had learned to weigh data and reject useless hypotheses, as seen in the reduction of total complexity between Fig. 3c and Fig. 4a, b, and c. A few students, however, seemed unable to do

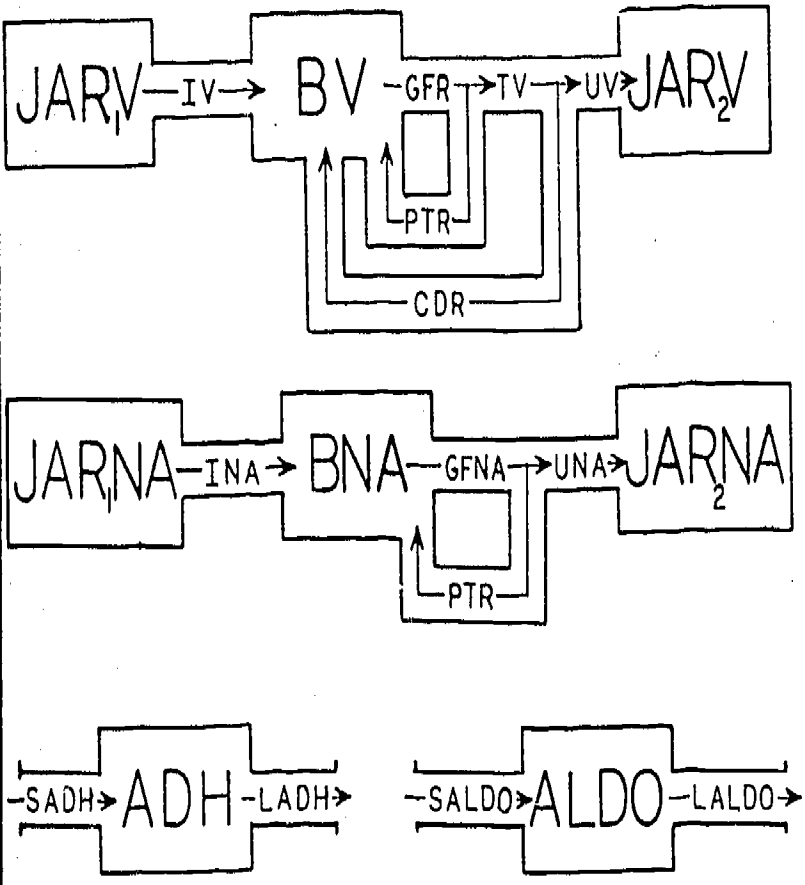


Fig. 1 A model of Kidney Function: Material Flow

BV = total blood volume
IV = rate of water intake
GFR = glomerular filtration
UV = urine production rate
BNA = total blood sodium
INA = rate of sodium intake
GFNA = tubular load of sodium
UNA = sodium excretion
ALDO = aldosterone level
SALDO = ALDO synthesis rate
LALDO = ALDO degradation rate
JAR V = volume in intake jar
JAR V = volume in collection jar
TV = tubular flow rate
PTR = isotonic reabsorption
CDR = water reabsorption
JAR NA = sodium in intake jar
JAR NA = sodium in collection jar
PTR = reabsorbed sodium
ADH = antidiuretic hormone level
SADH = ADH synthesis rate
LADH = ADH degradation rate

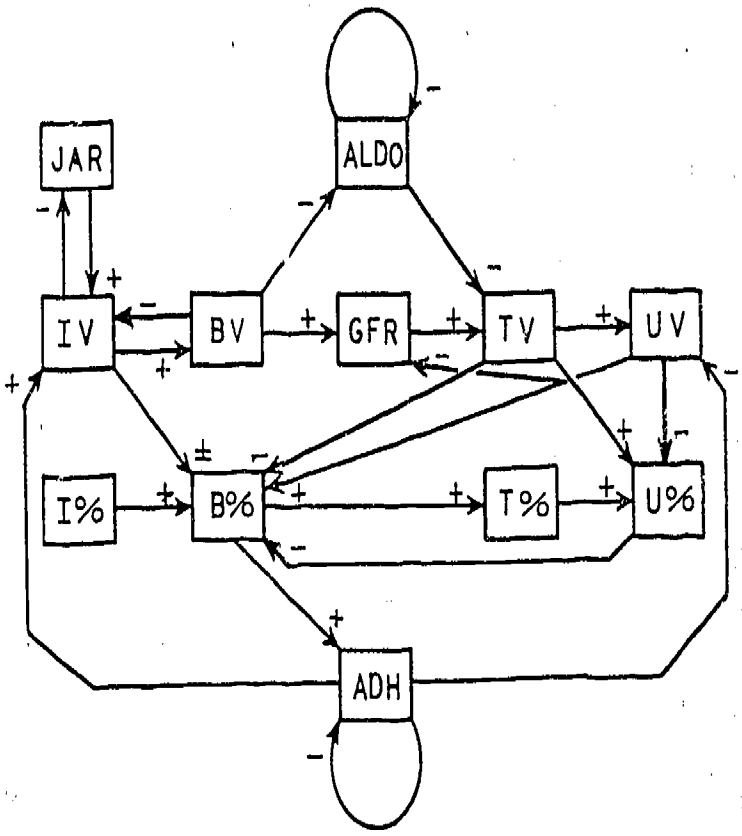


Fig. 2 A Model of Kidney Function: Signal Flow

BV = total blood volume
IV = rate of water intake
TV = tubular flow rate
B% = percent blood sodium
I% = percent sodium intake
ALDO = aldosterone level
JAR = volume in intake jar
GFR = glomerular filtration
UV = urine production rate
T% = percent tubular sodium
U% = percent urine sodium
ADH = antidiuretic hormone level

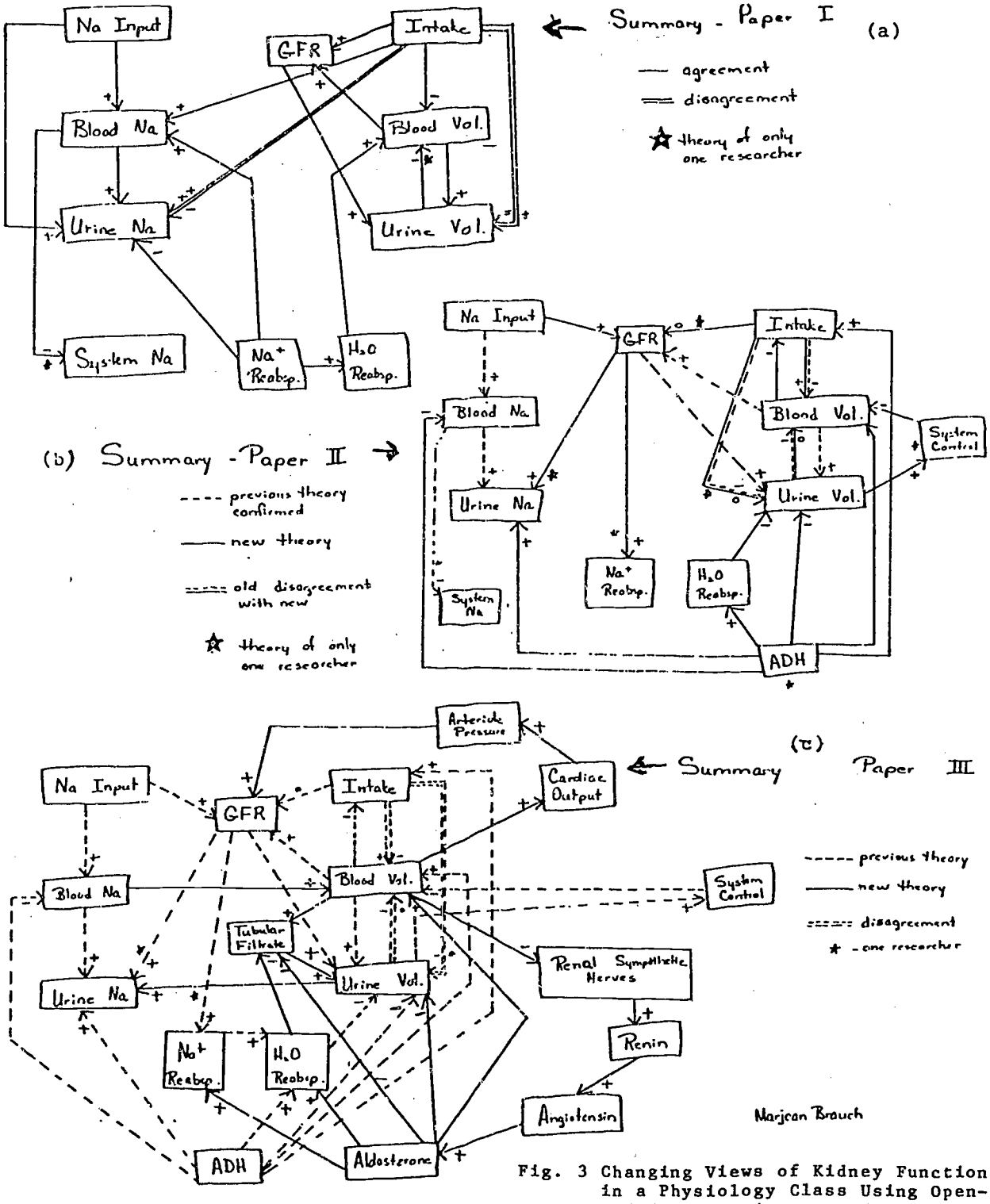


Fig. 3 Changing Views of Kidney Function in a Physiology Class Using Open-Ended Simulation to Deduce System Function. (Figures from student papers)

a. Summary of 10 student papers: Oct. 15
b. Summary of 10 student papers: Nov. 1
c. Summary of 10 student papers: Nov. 21

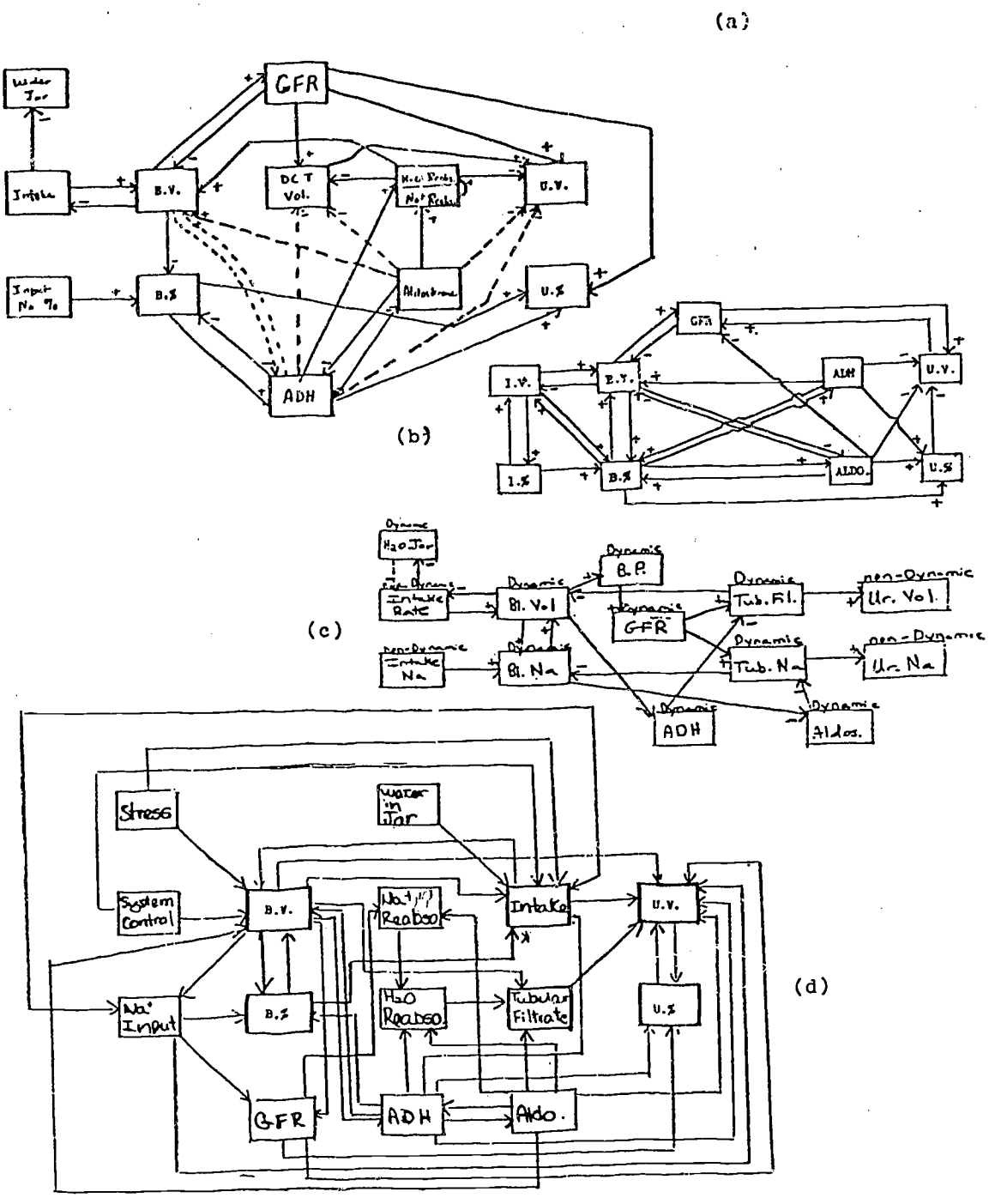


Fig. 4 Four Final Views of Kidney Function in a Physiology Class Using Open-Ended Simulation to Deduce System Function. (Figures from student papers)

this (Fig. 4d), retaining most of the postulated relationships of Fig. 3c.

Although the class developed a general consensus concerning the model kidney, areas of disagreement still existed at the time we terminated the project with a discussion of the real model (Fig. 2). As an example, consider the four views of the direct control of ADH expressed in Fig. 4. One student (Fig. 4b) was correct, one added two non-existant or remote relationships (Fig. 4a), and two of them (Fig. 4c and d) missed the direct link entirely. If the project had continued, this disagreement would have led to debate and research aimed at discovering the roles of blood volume and blood sodium in triggering ADH release.

The last observation is of great importance for evaluating the use of simulations as student research systems. By the end of the semester, more than forty individual investigations had been reported without completely explicating the dynamics of the model system. In fact, the class never moved beyond signal flow diagrams to the next logical step, writing their own model equations.

Model systems of the complexity of the kidney model (six differential equations) are very unlikely to be exhausted by student research. Such complex models will be as opaque to the student's understanding as the biological systems on which they are based, and if accurate, will be equally beneficial in teaching the dynamics of the system. In fact, since the student will be able to observe at least ten fold more events, many of them impossible in a student laboratory due to constraints of budget, time, equipment, and technique, a well designed and accurate simulation could be an even more effective experimental system than the real thing.

Conclusions

This report demonstrates how a simulation of a biological system, such as osmoregulation, may be effectively used in the teaching of physiology as a system for open ended student research projects. Students were able to conduct significant research as part of the course laboratory, and they were able to analyse their data and synthesize their new experimental knowledge into significant scientific models. In the process they evaluated experimental results which would have been impossible for undergraduates under traditional laboratory conditions.

Beyond the limited use of computers in specific disciplines such as Physiology, I believe this study may also have significance to a wider frame of reference, i.e., the whole process of science education. As our knowledge of the universe increases exponentially, it becomes increasingly difficult to train new scientists. Equipment costs more, the "core" information is far greater, more connections between disciplines are evidently necessary, and our educational resources are reduced. As a result, students today are unlikely to be able to use the tools of their discipline at their own discretion until they complete their terminal degree.

These problems can be met in part by the judicious use of computer simulation. Computers may simulate expensive equipment, both training students in its use and simulating its output for student analysis. Computers may also effectively demonstrate complex phenomena, including cross disciplinary topics, which include processes happening in time dimensions impossible for the laboratory. In addition, computers may give the student looking for a career a taste of the investigative process, and perhaps attract them into science. Last, but surely not least in the America of the 1980's, all this may be done at a reasonable cost. There is no need to use a large main-frame computer for effective student oriented simulation. With care, even the "home" microcomputers can produce surprisingly sophisticated results.

Bibliography

1. Postlethwait, S. N. "Improvement of Science Teaching." Bioscience 30:601-604; September 1980.
2. Anderson, David E. "Computer Simulations in the Psychology Laboratory." Simulation & Games 13:13-36; March 1982.
3. Wallach, Michael A. "Tests Tell Us Little About Talent." American Scientist 64:57-63; January 1976.
4. Kuhn, Thomas S. The Structure of Scientific Revolutions The University of Chicago Press, Chicago: 1970.

Appendix

Model of The Control of System Osmolarity

1. Equations of water balance

Jar = Jar - IV
IV = Mi / (1 + K1 / ADH + BV / K2)
GFR = TPR / BV
TV = GFR / (1 + ALDO / K3) x 0.8
BV = BV + IV - UV + Met
UV = TV (ADH / C1 + K4) / (ADH + K4)

2. Equations of salt balance

INa = 1% x IV
BNa = BNa + INa - UNa
UNa = TNa = T% x TV
T% = B% / C2
U% = UNa / UV
B% = BNa /BV

3. Equations of hormone level

ADH = ADH + Sadh - Ladh
ALDO = ALDO + Saldo - Laldo
Sadh = Mh /(1 + (K6 / B%) E10)
Saldo = M1 / (1 + (BV / K5)E10)
Ladh = ADH x Lh
Laldo = ALDO x L)

Physiology Simulation Usage

Semester	Course	Stdt.	Hrs.	Hrs/Stdt.
Fall 1979	Gen. Phys.	10	120	12
Fall 1979	Human P&AI	22	110	5
Spring 1980	Human P&AII	20	140	7
1979-1980	3 courses	52	370	7
Fall 1980	Human P&A I	17	54	3.2
Spring1981	Human P&AII	17	142	8.4
Spring 1981	Gen. Phys.	11	278	25.3
1980-1981	3 courses	45	474	10.5
1979-1981	6 courses	97	844	8.7

Parameters of Osmoregulation Model

Symbol	Value	Definition
IV	20	Intake rate
Mi	150	Maximum intake rate
ADH	100	Unite of ADH in blood
BV	1000	Blood Volume
Jar	1000	Amount in the water jar
TV	120	D.C.T. flow rate
TPR	0.2	Total peripherial resistance
ALDO	100	Blood level of Aldersterone
GFR	160	Glomerular filtration rate
UV	20	Urine production rate
U%	0.05	Meq. Na of urine
B%	0.1	Meq. Na of blood
T%	0.03	Meq. Na of tubular fluid
I%	0.05	Meq. Na of intake fluid
K1	150	ADH level at which IV=Mi/2
K2	500	BV level at which IV=Mi/2
K3	17.65	ALDO level:GFR/2 returns to BV
C2	12	Conc. effect: collecting duct
K4	175	ADH level: TV/2 returns to BV
C1	3	Dilution: Distal conv. tubule
L1	0.75	% loss of ALDO per hour
M1	150	Maximum release of ALDO
K5	1000	BV level: ALDO release=M1/2
E1	10	Intensifier of ALAO response
Lh	0.75	% loss of ADH per hour
Mh	150	Maximum release of ADH
K6	0.1	B% level: ADH release = Mh/2
E2	10	Intensifier of ADH response
INa	2	Total meq of Na in intake
BNa	100	Total meq of Na in blood
GNa	16	Total meq of Na in GFR
TNa	3.6	Total meq of Na in tubule
UNa	2	Total meq of Na in urine
Riso		Reabsorption: P.C.T.
RH20		Reabsorption: Collecting duct
RNa		Reabsorption of Na (total)
BP	100	Mean blood pressure
UJar	0	Volume: urine collection jar

MICROCOMPUTER-BASED DATA ACQUISITION FOR NEUROBIOLOGY

by Richard F. Olivo

Department of Biological Sciences, Smith College
Northampton, Massachusetts 01063

Abstract

A microcomputer system for capturing transient analog signals and displaying them repetitively on an oscilloscope has been developed for student use. The hardware is based on Rockwell's AIM-65, supplemental memory, and an analog-digital interface that is described. The software, in ROM, uses single keystroke commands, any of which may be entered at any time. Input modes include continuous input, triggered input, averaging, and rate histograms. Display modes include scrolling and jumping, both of which can be frozen or have their direction reversed, and slow output to a chart recorder. Intervals between samples are resettable in the range from 100 usec to 65 msec. The system has been used for three years; students have found it helpful, consistent, and easy to control.

Introduction

Many traditional laboratory courses can be made slightly easier or a bit more productive through the use of microcomputers, but a neurobiology laboratory can be substantially transformed by computerization. Neurobiology students typically record sequences of action potentials from nerve cells. Each action potential lasts only 1 or 2 milliseconds, although the sequence of events may extend over several seconds. The action potentials, amplified and displayed on an oscilloscope screen, seem to flash by; they can be glimpsed but not studied. They are too fast to be written by chart recorders, and although photography can capture them, photography is too inconvenient and too expensive for routine use in a student lab. A microcomputer with an analog/digital interface, however, can record these transient events for repeated playback to an oscilloscope, freezing the signal on the screen, or for slow playback to a chart recorder, to provide a permanent record. Several years ago, I reported on our preliminary plans to use microcomputers in an undergraduate neurophysiology course [1]; since that time, we have used the system for three years, the analog interface and the software have each been redesigned, and I can report now on what I regard as a tested and effective system for computerized data-acquisition. Although our system was developed for neurophysiology, it can be used for any data-acquisition task in which the sampling interval is between 100 microseconds and 65 milliseconds; this encompasses a wide range of laboratory applications. I would be pleased to make the software available to colleges or universities that wish to duplicate our data-aquisition system.

Hardware

At the time I designed our system, Rockwell's AIM-65 microcomputer was the most cost-effective choice for our purposes. We wanted six set-ups, one for each student group in laboratory, and thus the cost of each one was very important. Although a number of low-cost microcomputers are now available, the AIM-65 remains a good choice because of its particular combination of features. It includes a full keyboard for entering commands, and a 20-character display for prompts to the user; it has two input-output ports, for connecting the analog interface; and it includes a crystal-controlled 1-microsecond clock and 16-bit interrupt timer, for accurate timing of data-acquisition intervals. In addition, the AIM is compact and does not require a video monitor, which makes it easy to integrate into an existing laboratory set-up. The AIM's major disadvantage by current standards is that it has only 4K bytes of on-board memory, and expansion memory boards from various vendors are slightly more expensive than they ought to be. Nevertheless, a complete system (AIM, 32K expansion memory, and analog interface) costs about $1000, which is much less than systems based on other popular microcomputers and very much less than commercially available

data-loggers or digital oscilloscopes.

Although the AIM, enclosures, power supplies and expansion memory all are available from commercial vendors (some of whom will even assemble and test the package of components before shipping), there is not (so far as I know) a commercially available analog/digital interface board that is inexpensive, provides preamplification, and is capable of fast analog/digital conversions. We have built our own interface, based on a handful of integrated circuits that are relatively easy to use. My design for that interface, which incorporates improvements suggested by Artner Chace of Mount Holyoke College, is shown in Figure 1. The task of an analog/digital interface is to convert voltages to their digital equivalent, which is sent to an input port of the microcomputer, and to convert digital information from the computer back into voltages for display on an oscilloscope screen. In addition, because voltage signals from neurophysiological experiments are very small (even after preamplification, they are in the range of 10-100 millivolts), our interface includes a stage of preamplification to bring the input signals up to the size that the analog-to-digital converter expects (10 V); also, because we need to take in trigger signals from stimulators and emit pulses to synchronize oscilloscopes, the interface board includes connections and protection circuitry for trigger and pulse signals.

For analog/digital conversion, I chose integrated circuits from Analog Devices, Inc. (Box 280, Norwood MA 02062) that were relatively inexpensive, performed well, and required a minimum of external components. The analog-to-digital converter (AD 570) accepts signals in the +5 V to -5V range, and takes 25 microseconds to perform a conversion. To boost input signals to the required range, I added a two-stage amplifier based on the commonly available 1458 (dual 741) op amp. The amplifier has a variable-gain control (1x to 1000x in 1,2,5 steps) using a 10-position switch and a set of common resistors (these and other parts can be purchased from suppliers such as Radio Shack or Jameco). If calibrated gain is not needed, a 1-megohm potentiometer may be substituted for the selector switch. The preamplifier also provides a choice of direct- or capacitor- coupling (AC/DC switch), and it has an offset (zero) control; again, these may be omitted for ease of construction if they are not needed.

The output (digital-to-analog) conversion is provided by an AD558 integrated circuit, which produces voltages from 0 to 10 V and settles to its new output value within 2 microseconds after it receives the data. Note that the input circuit's range is -5 to +5V, while the output range is 0 to +10V. For each, the span is 10V, but the midpoint (the equivalent of 0 V at the input) differs by 5 V. This means that the analog output will be stepped up on the oscilloscope screen compared to the input. This has not proven to be a problem for students, but one could add a 1x amplification stage that offsets the output signal by 5 V to give the output the same midpoint as the input signal. The remaining components, the trigger-in and pulse-out lines, are protected from inadvertent connection to large signals by a comparator (LM 311) and by a resistor - diode network, respectively. The microcomputer expects to see (and produces) TTL-compatible pulses (0 or +5V), and if these are available from the other laboratory equipment -- and if the students are always careful -- the protection circuitry could be omitted.

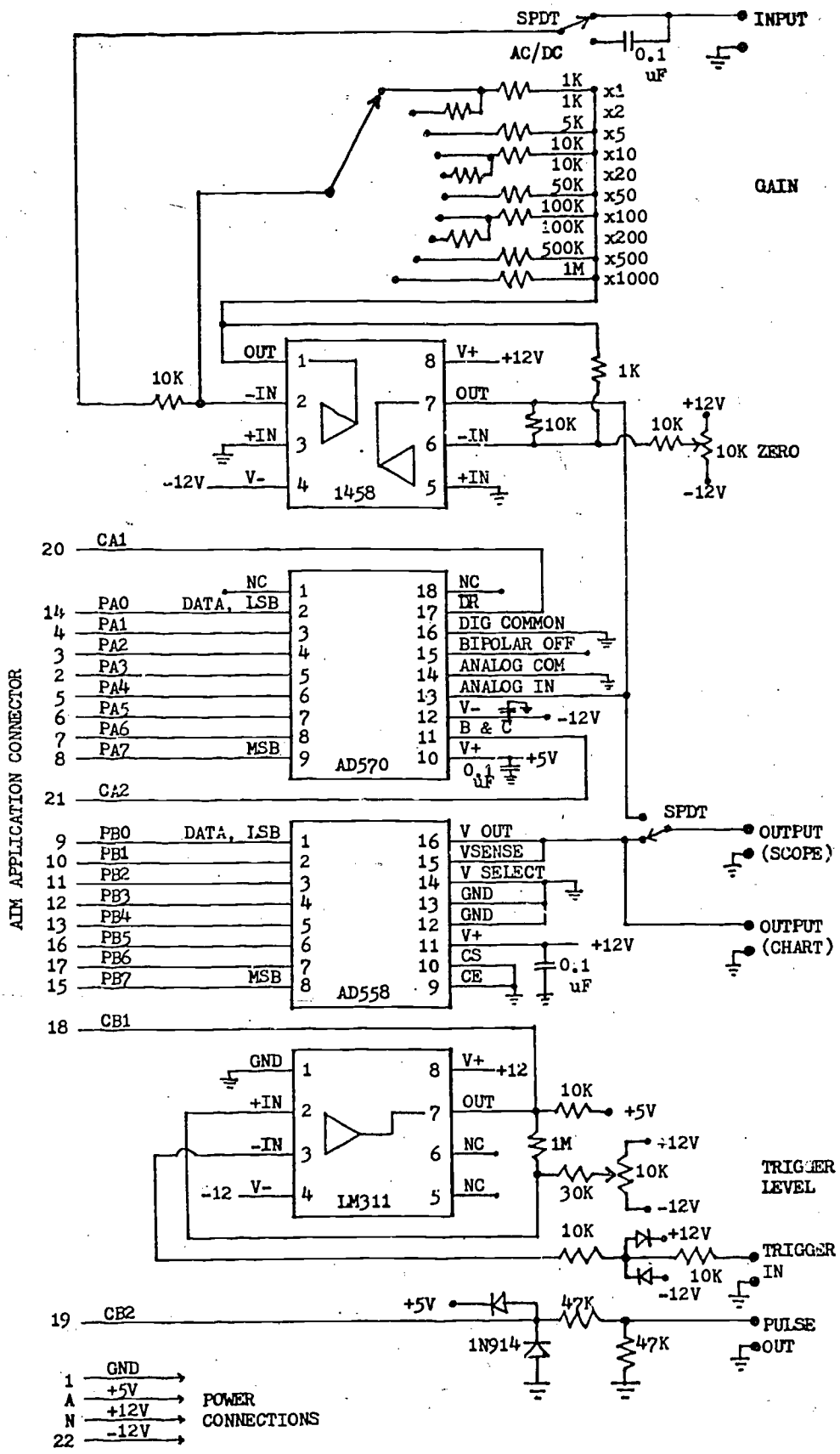
Although it generally is good advice to buy rather than build equipment, that option was not available for this system. However, since the two converters from Analog Devices are so easy to use, there is little difference between wiring a cable between the AIM's application connector and a board containing these circuits, and wiring a cable to an equivalent commercial board (if one were available). The additional components (preamplifier and trigger protection circuitry) are necessary for our application, but they may not be for yours; if they were omitted, the interface would be extremely simple to construct. The total cost of the integrated circuits is less than $40; our complete interface, which is mounted on a rack panel under the oscilloscope and includes the various features shown in the diagram, costs well under $100.

Software

The software for the data-acquisition system had to meet two principal criteria: it had to be fast, and it had to make the system easy to use for students who had no experience with computers. For ease of

---

Figure 1 (next page). Analog/digital interface based on integrated circuits for analog-to-digital (AD570) and digital-to-analog (AD558) conversion.


SPDT
INPUT
AC/DC
0.1 uF
1K x1
1K x2
5K x5
10K x10
10K x20
50K x50
100K x100
100K x200
500K x500
1M x1000
GAIN
OUT
V+ +12V
1K
10K
-IN
OUT
+IN
-IN
10K
+12V
10K
10K ZERO
-12V
+IN
-12V V-
1458
CA1
NC
NC
PA0 DATA, LSB
DR
PA1
DIG COMMON
PA2
BIPOLAR OFF
PA3
ANALOG COM
PA4
ANALOG IN
PA5
V-
-12V
PA6
B & C
PA7 MSB
V+
AD570
0.1 uF
+5V
CA2
SPDT
PB0 DATA, LSB
V OUT
OUTPUT (SCOPE)
PB1
VSENSE
PB2
V SELECT
PB3
GND
PB4
GND
OUTPUT (CHART)
PB5
V+
+12V
PB6
CS
PB7 MSB
CE
0.1 uF
AD558
CB1
GND
V+ +12
+IN
OUT
10K
+5V
-IN
NC
1M
+12V
10K
TRIGGER LEVEL
-12 V-
LM311
NC
30K
-12V
10K
+12V
TRIGGER IN
10K
-12V
+5V
47K
CB2
PULSE OUT
1N914
47K
GND
+5V
+12V
-12V
POWER CONNECTIONS
AIM APPLICATION CONNECTOR

use, the program is stored in permanent memory (ROM), so that no loading from disk or tape is necessary, and no accidental over-writing of the program can occur. Program start-up and all subsequent commands are by single keystrokes, and any command may be executed at any time. The command set is listed in the Appendix, and will be discussed further below. Each student receives a User's Guide, which explains each command. The cover of that Guide (Figure 2) also illustrates the performance of the system; the photograph

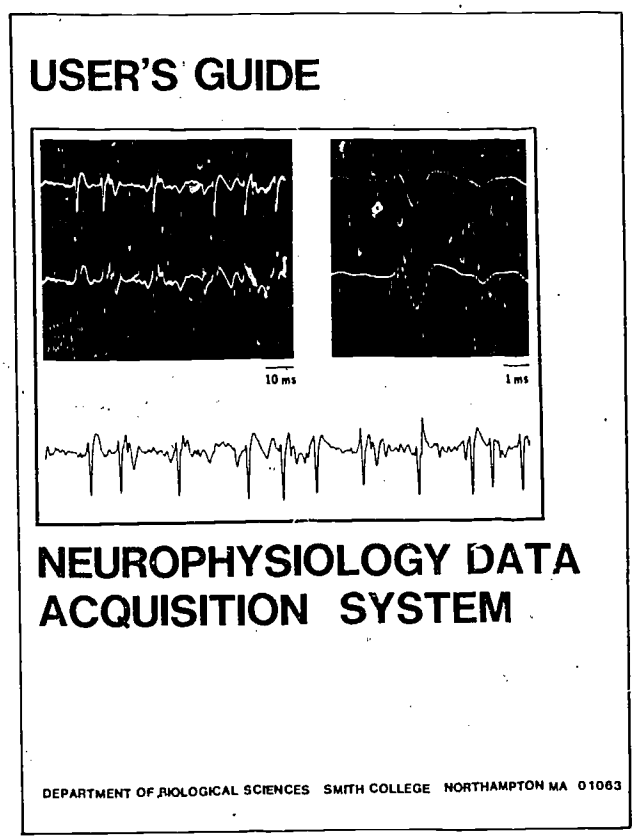


Figure 2

at the top left shows an example of analog data (upper trace; the brief downward spikes are action potentials) and the digitized echo of the same data from the computer (lower trace). At the right, a similar photograph of the oscilloscope screen shows the original and digitized data at a faster sweep speed. The chart record below was obtained by playing back the same data from memory at a slow rate; note the accurate reproduction of the analog signal.

The data-acquisition program is written in assembly language, and is contained in a 4K ROM that plugs into a socket on the AIM intended for Rockwell's assembler. Although writing an assembly language program of this size is a substantial task, it was clear to me that the data-acquisition routines would have to run at machine speed (eliminating the possibility of working in BASIC), and it seemed simpler to do everything in assembly language rather than trying to mix high-level and assembly language. I have described elsewhere [2] an example of an assembly language routine for acquiring one byte of data. In retrospect, now that FORTH is available for the AIM, I would probably write the program in FORTH (which makes it easy to include assembly language routines) if I were to write it again.

The program has three major input modes: continuous input, triggered input, and averaging. These place data into three separate buffers in memory. At start-up, the available memory is automatically allocated into three non-overlapping buffers, but the allocation may be changed by using the VARIABLES command, and any buffer may be assigned to any part of physical memory. Our systems have 36K of memory, which means any buffer could store up to 36,000 samples. In continuous INPUT mode, the input buffer is filled repeatedly (with the newest data over-writing the oldest) until INPUT is halted or another mode is selected. Each digitized data point is echoed to the oscilloscope as it is taken. INPUT mode is useful for capturing spontaneous neural activity and responses to hand-delivered stimuli (such as mechanical prods or the application of drugs). When an event of interest has occurred, one halts INPUT mode and displays the buffer, which contains the last few seconds of data. When electrically triggered stimuli are used, however, TRIGGER mode provides a more efficient way of capturing the response evoked by the stimulus. In TRIGGER mode, the computer waits for a trigger pulse; when the pulse is received, a pre-set number of data samples is taken and stored, after which the computer awaits the next trigger pulse. The amount of data stored in each sweep is determined by a variable, which can be reset in units of 256 samples (1 page of memory); the default value is 2 pages (512 samples).

The entire orientation of the system is toward temporary storage of data, with the expectation that data of interest will be written out to a chart recorder for a permanent copy. This gives students their data in its most useful form for further analysis, and it also avoids the expense

and complexity of disk or tape (either of which, from my point of view, only postpones the moment when a hard copy must be made). Data that have been captured by the system can be played back by typing the DISPLAY command, D. DISPLAY automatically selects SCROLL mode if continuous input was used, and JUMP mode if triggered input or averaging was used. In addition, SCROLL or JUMP may be entered directly (and any buffer examined) by typing S or J.

SCROLL is a particularly useful display mode. The signal appears continuously on the oscilloscope screen but seems to slide across it, like a moving chart. The direction of travel can be changed by typing R, REVERSE (this is a feature that was added to the system at student request); the image can be frozen (or scrolling resumed) by pressing the space bar; and the rate of scrolling can be increased or decreased by pressing the + or - keys. As a result, one can examine data with very high temporal resolution (our oscilloscope sweep speed is typically 2 ms/div), move forward or backward in time at one's convenience, freeze time, expand the oscilloscope sweep to examine events of interest more closely, and handle a large data-sample.

JUMP mode is the display mode selected automatically if triggered input or averaging was the last input mode used. One sweep appears frozen on the screen, and then the system JUMPs to the next sweep. Jumping can be halted (or resumed) by pressing the space bar, and the direction of advance can be reversed by typing R. The current sweep number is shown on the AIM's display, an asterisk appears if jumping has been halted, and an R appears if jumping is reversed. Once again, by freezing a sweep of interest on the screen, experimental data can be examined at length, the sweep can be expanded, one can back up to the previous sweep, and so forth.

In addition to these display modes for an oscilloscope, two other display modes produce slow output for a chart recorder. EXCERPT (E) plays out the segment of memory currently displayed on the screen. WRITE plays out the whole buffer, starting from the beginning. Either of these can be used for making a chart record of data obtained from any input mode. For convenience, the rate of sending out data samples is slower for EXCERPT than for WRITE, so that one chart speed can be used to produce a high-resolution (E) record or a temporally compressed (W) overview. However, the default value of any timer interval can be changed using the VARIABLE command. For our work, fast acquisition, fast oscilloscope display, and slow output to a chart recorder make the most sense, but the software is written in such a way that it can also be used for acquiring slow data (approximately 16 samples per second, minimum), where it might be appropriate to use a display rate much faster than the input rate, and a chart writing rate that is equal to the input rate.

Finally, the two special input modes deserve some comment. AVERAGING resembles triggered input in waiting for a trigger pulse before acquiring a sweep of data, but the data are summed with previous sweeps rather than stored separately. The average is calculated whenever the number of sweeps is an exact power of 2 (1, 2, 4, 8,... 256). At each sweep, the last calculated average (rather than the input signal) is displayed, while the AIM shows the number of the current sweep and the number of sweeps in the last average. Averaging is useful for extracting an evoked response that is hidden in a noisy baseline, and students have used it more often than I first expected. During averaging, DISPLAY can be entered to show the current average frozen on the screen, and averaging can then be re-entered without losing the accumulated data. The second special input mode, HISTOGRAM, is probably of more interest to neuroscientists than to scientists in general. HISTOGRAM sends to a chart recorder a voltage proportional to the number of events per second, updated in 0.1-sec bins. It is used for counting the number of action potentials per second, an important measure of neural activity. In this implementation, events are detected in software: any signal that crosses the 0-volt baseline (either in a positive or negative direction, as set by the user) is counted as an event, and the count is updated every 100 msec. By using the zero offset control, it is usually possible to adjust an input signal so that only the biggest events trigger the counter. A pulse is emitted whenever an event is detected; one can trigger the oscilloscope from the pulse to observe the input signal being counted, while the chart recorder simultaneously writes a record of the histogram.

Evaluation

The data-acquisition system has been used in my neurophysiology course for three years by undergraduates whose prior experience with computers ranged from extensive to none. Even the least experienced students found the computer no harder to use than the other equipment they encountered, and all of them were eager to use the system since it let them

see their data clearly and gave them hard copies to show others. The command set seemed consistent and easy to understand; the students controlled the system properly right from the start, and readily made it do what they wanted it to do. As a result, while previously students in the course had at best obtained a few polaroid photographs from some experiments, they now routinely capture as much of their data as they wish. They work more quantitatively, and the level of the laboratory now is better matched to their other work in the course.

References

1. Olivo, R. F. (1980) Microcomputers as laboratory instruments: two applications in neurobiology. Proc. National Educ. Computing Conf. 2: 81-85.

2. Olivo, R. F. (1981) An efficient A/D interface. Compute! 3 (9): 140-142 (September 1981).

Appendix

SUMMARY OF DATA-ACQUISTION COMMANDS

When the computer is first turned on:

N starts the data-acquistion program and sets default values. All subsequent operations are controlled by the following commands:

Input modes

I Input. Continuous storage of data in the input buffer. New samples overwrite the oldest previous samples.

T Triggered input. A single sweep of data is stored after each trigger pulse is received. Data are stored in the trigger buffer.

A Averaging. Successive sweeps of data, initiated by trigger pulses, are averaged. The last average calculated is displayed.

H Histogram. Events that cross ground level (with + or - slope, as specified) are counted, and the count is output as a voltage every 100 msec. No data are stored.

Output modes

D Display. Automatically selects the appropriate output mode and buffer, determined by the last input mode that was used.

S Scroll. Data from the buffer selected scroll across the screen, as on a chart recorder. Usually used with the input buffer.

J Jump. Successive sweeps from the buffer selected appear one after another. Usually used with the trigger buffer.

W Write. Contents of a buffer are output slowly, for writing by a chart recorder. Outputs the entire buffer.

E Excerpt. The part of a buffer currently displayed on the oscilloscope screen is output very slowly, for writing by a chart recorder.

C Continue. Resume last display mode from the current position; used after Excerpt.

System control

Q Quit. Exit from the current mode; this command is available at all times.

(space) Pause. On input modes, suspend data collection. On output modes, suspend advancing through the buffer, freezing the current data on the screen. Typing (space) again ends the pause.

R Reverse. Change the current direction of scrolling or jumping.

+/- Increase or decrease the rate of scrolling or jumping. May be typed more than once.

V Variables. Print current values for memory allocations, timer intervals, or constants, and permit their alteration.

COMPUTER LITERACY

Margaret Christensen
Dr. Carla Thompson
Dr. Joyce Friske
William H. Pritchard, Jr.
Donald Z. Spicer
Ronald R. Bearwald
David J. Lewis

ABSTRACT: Algebra, BASIC, and Computers: The ABC's for Non-Science Majors

Margaret Christensen, Widener University

Seven realistic problems lead students through a semester of elementary algebra and computer science in a new course we have developed.

As computers become less the exclusive domain of the scientist or specialist and more the daily companion of educators, social scientists, humanists, and businessmen alike, the pressure to become computer literate increases for all students, including those who lack the mathematics proficiency which is prerequisite to successful completion of computer courses. Rather than subject these students either to a computer course which is devoid of math (if such is possible!) or to a traditional, remedial college math course taught in the same way that lost them before, we have developed a course which combines computer science and elementary algebra in a symbiotic relationship.

Central to the course are seven problems which require some thought, the use of algebra, and computer programming in BASIC for their solution. In the course manual the problems sections follow a logical order of progression in terms of material covered and difficulty, and have solutions which require only the use of material taught in that section or in previous sections of the course.

Students have responded enthusiastically; they have clearly learned a lot; and the instructor found the course a lot more fun to teach than standard remedial algebra or computers without math.

ABSTRACT: Computer Literacy in the Two-Year College Curriculum

Dr. Carla Thompson, Dr. Joyce Friske, Mathematics Instructors, Tulsa Junior College - N.E. Campus, 3727 E. Apache, Tulsa, OK 74115

As the belief in the need to educate future citizens in the operation, use, and impact of computers gains support in educational circles, it is becoming increasingly clear that all students should be provided with educational opportunities that allow them to become computer literate. This need can be appropriately met at the junior college level by offering computer literacy courses designed for non-computer science majors.

This presentation will focus on the development and integration of a computer literacy discipline area (CLT) into the two-year college curriculum. Presenters will describe the role of the CLT discipline area within the overall curriculum and discuss the focus of CLT courses with respect to specific needs of two-year college students. Concerns associated with integrating computer literacy couses into a non-computer science division will also be discussed.

In order to meet the computer literacy needs of all students regardless of background or major, the CLT curriculum was designed relative to three areas: liberal arts students, technical/occupational students, and education majors. Course descriptions, laboratory activities, and class projects will be suggested for all computer literacy courses within the CLT curriculum.

The session will conclude with an overview of future perspectives for the two-year college computer literacy scope and sequence.

ABSTRACT: The Vassar College Computer Literacy Program

William H. Pritchard, Coordinator, Computer Literacy Program, Donald Z. Spicer, Associate Dean of the College, Vassar College, Poughkeepsie, NY 12601

In January 1982, Vassar College with support from the Fund for the Improvement of Postsecondary Education (FIPSE) initiated a computer literacy program that has a number of unique features that resolve many problems associated with computer literacy in the context of a relatively small liberal arts college. The thrust of the program is to infiltrate computer usage into the curriculum of the College as a whole. Therefore faculty development, which provides sustained impact, is an important component. In a core course a group of participating faculty, together with students sharing common interests, are introduced to the availability and use of computer resources that support their discipline. To reinforce this initial experience, participating faculty agree to introduce computer usage into designated courses taught in later semesters, and students who take the core course are expected to also take at least one of those subsequent courses. While many aspects of educational computing can be managed on micro and minicomputers, there are also many uses

required to support diverse faculty interests that cannot be cost efectively acquired by a small college. Since the Vassar Computer Literacy Program is intended to support a broad cross-section of the faculty, the program makes use of the varied software and hardware resources available remotely through EDUNET. Data base and information services such as DIALOG, Compuserve, and The Source are also used.

A major aspect of the Vassar program is that it is designed to be a model program that is intended to be readily transportable to other colleges. Therefore, the program is designed to have relatively modest start-up costs. The major continuing costs are for personnel and remote teleprocessing.

Beginning Febraury 1893, the program will be expaned to include more courses, an increased number of faculty development workshops, and a unique program of faculty ownership of microcomputers. Under this program faculty will be encouraged to purchase their own microcomputer through joint support from a grant from the Sloan Foundation, a College low-interest loan, and a negotiated group buying plan with a vendor. It is hoped that within the first year, approximately 25% of the faculty will own a microcomputer.

The presentation in this session will elaborate on the initial conception of the program, explain the design of the core course, and discuss the experience learned during the first year and a half of implementation.

ABSTRACT: A Microcomputer Literacy Program

Ronald R. Bearwald, Assistant Superintendent, Lincolnwood Scool District Number 74, 6950 East Price, Lincolnwood, IL 60645

The Lincolnwood Schools have developed a plan for mocrocomputer literacy which provides for the instruction of every child in grades two (primary), four (intermediate), and six (upper). This instruction will be based on a sequential curriculum which includes general cognitive and affective goals as well as specific operational and programming objectives.

General goal of the microcomputer literacy program are:

- To develop computer literacy by teaching important computer related concepts, increasing the awareness of the values and applications of computers in our world, and providing opportunities to attain a certain level of competency in performing fundamental computer operations.
- To improve the overall cognitive and affective abilities of each child by developing structured and logical thinking skills, positive attitudes, and innate creativity.

Each student in the specifieed grades will receive a minimum of four weeks instruction which will take place in the following manner:

Grade 2: Students will come to the micro learning area in groups of six. Each student will receive four weeks of instruction; two consecutive weeks during the first semester and two during the second. Classes will be twenty-five minutes in duration. The computer language used will be LOGO.

Grade 4: Students will come to the micro learning center in groups of twelve. Each student will receive four consecutive weeks of instruction. Classes will be forty minutes in duration. The computer language used will be PILOT.

Grade 6: Students will come to the micro learning center in groups of sixteen. Each student will receive four consecutive weeks of instruction. Classes will be forty-five minutes in duration. The computer language used will be BASIC.

Students will be grouped heterogeneously for instruction. Additional opportunities will be provided to students in the non-target grades on an individual, small group, and special-unit basis. All students will have equal access to participation regardless of ability level.

ABSTRACT: Machine Language in Computer Literacy: Strategy and Supporting Software

David J. Lewis, Department of Mathematics, Ithaca College, Ithaca, NY 14850

A grasp of fundamental principles of machine language is an important subject for computer literacy in its own right. It can also serve as a concrete introduction to other topics such as variables, the CPU, main memory, hardware vs. software, high-level language processors, the notion of a program itself, and many others. Further, machine language experience illuminates the otherwise murky gap between logic design and high-level language. For these reasons computer literacy courses at Ithaca College generally begin with one to two weeks on Slic, a highly interactive, full-screen, instructional simulator for a simplified machine language we have developed for the purpose to run on the Apple II.

Slic is a simple decimal machine like those in many computer literacy and data processing texts. What is important about the Slic processor is its totally visible memory and animation of program execution under student control. The accumulator, program counter, all seventy memory locations, and a display/reply "screen" remain on the Apple screen, reflecting all events in the execution of a program in "real-time".

Slic renders many concepts graphic that students must ordinarliy infer, even with the best of the available high-level language processors. The Slic intruction being executed is highlighted in reverse video, and one can SEE a loop in action as the program runs and a sum grows in the accumulator. The instructor can POINT AT a variable (that is, a location) on the screen and change it by executing an instruction. Students can discover the principles of destructive read-in and non-destructive read-out simply by observation. The sharing of main memory by program and data appears natural when seen in real-time action.

The major technical features of Slic are: (1)

target machine instructions and data are stored in five-digit decimal locations and a single accumulator; (2) the instruction repertoire includes laod and store, arithmetic, jumps, and input/output; (3) the command structure is based on the UCSD P-system and Apple Pascal; (4) a memory "editor" allows direct manipulation of "main memory" and easy entry of programs; (5) a filer provides saving, retrieval, and printing of programs; (6) a program may be run at a specified rate, stepped one instruction at a time, or "crawled" through all fetches and stores; (7) any machine instruction may be executed in immediate mode, simply by typing it as a command.

COMPUTER EDUCATION FOR ELEMENTARY SCHOOL TEACHERS

Joyce Currie Little
Robert Wall
Harold R. Strang
Ann Booker Loper
Dr. Alice-Ann Winner
E. Muriel J. Wright
Helen V. Coulson

ABSTRACT: Computer Literacy for Elementary and Middle School Teachers - A Report on a MAEUC/AEDS Project

Joyce Currie Little, College of Natural and Mathematical Sciences, Robert Wall, College of Education and Instructional Technology, Towson State University, Towson, MD 21204

A pilot project in computer literacy, sponsored by the Maryland Association for Educational Uses of Computers, Inc. (MAEUC), and the Maryland affiliate of the Association for Educational Data Systems (AEDS), was done to test a strategy for increasing literacy among teachers in elementary and middle schools. The primary purposes of the project were to provide information to the schools for awareness of computer literacy; experiment and evaluate the use of certain exercises, materials, and activities; determine computer literacy levels; and determine the effects of exposure to computer literacy materials. It was decided that a minicourse format would be used.

Three elementary schools and their feeder middle school in an inner city area were chosen for the minicourse. All teachers and administrators, and some parents, were invited to participate. The course was organized, managed, and taught by members of MAEUC/AEDS, and equipment was provided by their institutions.

The Computer Literacy and Assessment questionnaire, developed by the Minnesota Educational Computing Consortium (MECC), was used for pre and post testing of knowledge and attitudes in order to assess changes in computer literacy.

This presentation will describe the project and its management, give an overview of the minicourse, present the results of analysis of change, and summarize the local MAEUC/AEDS group's recommendations.

ABSTRACT: Microcomputer Simulation: An Aid in Training Elementary School Teachers

Harold R. Strang, Ann Booker Loper, 405 Emmet Street, Ruffner Hall, Unversity of Virginia, Charlottesville, VA 22903

This micocomputer project is being developed to assist in the preparation of elementary school teachers. Our primary goal is to help bridge the gap between academic teacher preparation and actual teacher-student classroom contact.

Hardware components consist of an OSI 48k microcomputer equipped with two 8-inch disk drives, a real time clock, a video terminal, a second independent video display, and a serially connected printer.

After loading the simutlation's BASIC program, an operator assigns, via the terminal's keyboard, the following to each of the class's four simulated students: a probability for knowledge (the likelihood of answering correctly) and a probability for initiative (the likelihood of volunteering to participate).

The teacher, who is situated in a booth physically isolated from the operator, next receives a brief familiarization with the simulation's graphic feedback characteristics and then actually conducts a lesson with the four students whose names are displayed on the independent video monitor in front of the teacher. The teacher may verbally instruct, ask questions, solicit questions or answers, or just respond to students whose graphically defined hands pop up due to programmed initiative. Each teacher-student interaction is coded by the operator and keyed into the terminal. The terminal's display directs the operator, who functions as the voice of the students, what to say to the teacher. After the teacher has solicited an answer, for example, the operator's screen might present WRONG ANSWER 1, the 1 designating the type of wrong answer to be relayed to the teacher. Rapid computer feedback to the operator coupled with careful operator training insure a pace in the teacher-student verbal exchange not appreciably slower than that found in an actual live classroom.

During the teaching session, the computer also collects time and frequency data on over 110 specific types of cycles that the teacher has employed in interacting with each student. As cycles emerge, the printer functions as an event recorder, displaying immediately following each cycle the student involved, the time elapsed, and the cycle type. Cycles are defined according to their antecedent characteristics (e.g., teacher soliciting an answer, a question, or responding to a 'hands up'), student responses (e.g., right answer, wrong answer, or question), and teacher consequation (e.g., inclusion of feedback, instruction, and/or emotional affect).

A complete hard copy teaching profile is obtained at the conclusion of the teaching sesion. This record displays over 500 separate measures pertaining to the events that occurred during that session.

Two projects have addressed one of the most

basic questions pertaining to the simulation - namely, whether it truly creates an environment functionally similar enough to an actual classroom to be of value in teacher training. Initial results are very encouraging. When, for example, class or individual characteristics were manipulated so as to produce initiative and/or knowledge contrasts, participating teachers responded in ways paralleling those that would be expected in actual classrooms.

In addition to pursuing the validation research, the authors are also exploring the possibility of developing a pre-teaching diagnostic tool based on the system's extensive data collecting, sorting, and display features.

ABSTRACT: Toward Curriculum Development: A Case Study in Computer Inservice Training

Dr. Alice-Ann Winner, United Nations International School, 24-50 East River Drive, New York, NY 10010

Teacher training in appropriate use of the microcomputer in the elementary classroom presents a problem for educators. Many of the computer training programs are conducted by "experts" who have little understanding of elementary education and/or the specific needs of its teachers, and consequently curriculum development for wise computer use has not been forthcoming. This project report describes the design, inplementation, and evaluation processes of an inservice training program especially created for elementary teachers.

Three other differences separate this training program from most of the current computer workshops, even though similarities in content and structure exist: 1) the use of an inside change agent as the inservice instructor; 2) the extended duration of the workshop series over the entire school year; 3)the use of formative evaluation in the design process.

The development of this program started with reviews of the literature on change and computing in education with regard to the implications on curriculum development in elementary education and inservice training. The training took place at a single school with an international population. While the demographics of the study were to some extent idiosyncratic, the course content is applicable to other school environments as many elementary teachers share common goals. The materials and experiences designed and implemented in this study reflect this commonality of purpose.

The objectives of the program were to increase the awareness, exploration, and experimentation levels of the participating teachers, and to lay a foundation for permanent computer implementation and curriculum revision. Designed by the author, who as a fellow teacher was aware of the regularities of the school environment, it reflected the specific needs of the participants. These needs were addressed in the workshop sessions and in the intervals between sessions. Use of formative evaluation permitted the teachers an active role in the planning process, transmitting a sense of shared responsibility.

Five assumptions were articulated and supported by the results of the study. Briefly stated they are: 1) workshops conducted by outside agents are less effective in promoting innovation adoption in elementary education; 2) a single school, which possesses a positive attitude toward change and the requisite supportive mechanisms, can be a decisive instrument of change; 3) an inside change agent can initiate a successful change process in a single school environment; 4) computer literacy at the introductory level needs to include careful analysis of "wise use of microcomputers" to avoid an over-emphasis on computer assisted instruction as the prominent mode of implementation; 5) computer literacy at the elementary level should provide an awareness of the implications of process learning to develop curricula for optimal implementation of the computing potential.

The data, collected from a variety of evaluatory procedures, support an additional assumption undefined at the outset - curriculum revision should follow rather than preceed teacher training. The results also indicated apparent stages in the computer implementation process, and illustrated some of the problems inherent in that implementation as well as general ones relating to inservice training. The data underscored the necessity of integrating developmental inservice training into the regularities of the elementary school.

ABSTRACT: Incorporating the Microcomputer into the Department of Mathematics Program for the Prospective Elementary School Teacher at California State University, Northridge

E. Muriel J. Wright, Helen V. Coulson, California State University, Northridge, Northridge, CA 91330

Computer literacy is an essential outcome of contemporary education. To keep pace with the inevitable sophistication of their students, it is imperative that K-8 elementary teachers - perhaps the single most influential group in the mathematical competence of the country - acquire an understanding of the versatility and limitations of the computer through a working knowledge of how one interacts with computers and how one uses their capacities.

The Department of Mathematics at California State University, Northridge, has just completed the PILOT MODE of a two year project for (1) the development of a laboratory of eight microcomputers, (2) incorporation of interactive computer experiences within an existing strong two semester mathematics program for prospective elementary school teaschers involving 700 students annually (90% women, 20-40% ethnic minorities, 10% re-entry women), (3) selection and design of software using the computer's special capabilities (iteration, recursion, graphics, number-crunching) to enhance the treatment of the mathematical topics within the available time, (4) bringing the novice to a beginning but sound programming capability.

We will (1) build mathematical and computer skills - by hands-on drill, testing, and assignment correction, (2) teach new mathematical and computer concepts - by tutorial programs which will also invite student modification, (3) provide problem solving experience - by interactive programs in simulation of probability, statistical, and transformation geometry problems, (4) illustrate by examples of our programs and format the variety of uses that an elementary teacher may make of the microcomputer in the classroom - in mathematics, but also in other aspects of the curriculum, and (5) develop programming skills - by teaching computer commands, by requiring modification of some existing programs, and by student-written programs suitable for elementary classroom use.

During the PILOT MODE, using an Apple microcomputer and peripherals purchased under a grant, we were able to select and develop software for drill, tutorials, and simulations; make extensive student testing of the materials in both one-unit electives and seminars; and determine the values of individual and paired use of the terminals for student work. A room for setting up a model microcomputer laboratory housing eight machines was obtained adjacent to the classroom/tutorial complex used for the methematics courses. Proposals were submitted to various agencies within and outside the University. Six Apples with color monitors and disk drives have been granted and will be installed during January 1983. Peripherals such as a graphics tablet and two printers will also be in use. A voice-over attachment for use with handicapped students is attached to one computer.

Beginning with the Spring Semester 1983, full implementation of the project is planned. Using tested materials - mathematics text, programming manual, computer film, and computer programs and materials selected and designed by the two project directors - three hundred students under ten instructors will develop programming capability along with the required mathmatical knowledge. For years, a standardized common final has been administered to all sections of this course so that well established norms are available for comparison of the effect of incorporating the microcomputer into the program without reduction of mathematical content. Level of Computer Literacy will be determined using a test and norms developed by the Minnesota Educational Computing Consortium. Finally, instructor and student attitude will be determined by self-report.

The goal is preparation of 350 students per semester trained in the mathematics essential for the elementary school and with a working knowledge of computers. An important spin-off will be the replacement of the now superfluous one-unit elective in introductory computer programming with a course in Graphics and Turtle Geometry.

C. Michael Levy
Dr. John R. Pancella
Edward Zeidman
Melvin H. Wolf

ABSTRACT: Real-Time Microcomputer Programs for Teaching Statistics

C. Michael Levy, Department of Psychology, University of Florida, Gainesville, FL 32611

While many instructors believe that the key to mastery of statistics is the successful completion of many sample problems, it is often infeasible for them to implement instructional programs that provide students with a rich corpus of problems. Partly in resonse to this implied need, I have developed a 100 kilobyte ensemble of interactive CAI exercises for the Apple, IBM, Atari, and DEC microcomputers. The materials are solely instructional; they provide no means to perform statistical calculations of user-supplied data. These programs are used in conjunction with a workbook that portrays the rationale and demonstrates the use of each statistic with a detailed example, and presents scenarios describing the problems that the student completes.

The design of our programs was heavily influenced by insights that emerged as I developed other courseware for undergraduates during the last decade. For example, the software contains routines whose sole function is to reduce the opportunities for students to make entry errors that result in abnormal program termination or evocation of an obscure error message.

The ensemble of programs contains two broad categories of exercises. One includes multiple computational problems for each statistic that is in the family of problems found at the end of the chapter of many statistics books. These were designed to provide the guided practice many students need to appreciate the process of determining the correct solution.

I have dubbed the second category of exercises "Exploratory Problems". These were designed expressly to give students an intuitive grasp of each statistical technique. Great effort was made to incorporate special features that would make moderately deep levels of cognitive processing nearly unavoidable. This was accomplished by directing students to "play with" the techniques and "manipulate" the raw data in order to answer "what if" sorts of questions - for example, "What will happen to the variance if I add a constant to each score?" "What happens to the standard error if I increase or decrease N?" "What will happen to the value of t or F if I reduce to zero the variance of one group?"

Graphics were used whenever possible to enliven the displays, and color was used not merely to enhance their visual quality but to serve as a signal or cue. More importantly, the keyboard was used as an asynchronous device, permitting us to create dynamically changing displays and, in general, to give these programs many of the characteristics of some of the exhilirating arcade games. The obvious appeal of this utilization is related to the fact that students who often resent using "canned" data feel that they (rather than the computer) are in primary control in these situations.

In short these programs provide students with powerful opportunities to discover for themselves the answers to important substantive questions, thereby making the abstract theoretical ideas of statistics more concrete and memorable. The presentation will include both a discussion and demonstration of these materials.

ABSTRACT: High School Science Microcomputer Project

Dr. John R. Pancella, Coordinator, Secondary Science, Mr. John Entwistle, Teacher Specialist, Science, Mathematics, and Microcomputers, Mrs. Carol Muscara, Specialist, Computer Related Instruction, Montgomery County Public Schools, Rockville, MD 20850

A project was begun during the 1981-82 school year to implement microcomputer technology in senior high school science courses. Feedback from three school tryout centers was used to develop teaching and training plans to incorporate microcomputers in biology, chemistry, earth science, and physics classes of 12 senior high schools during the 1982-83 school year. Training for 36 science teachers was conducted during summer 1982. Use of microcomputers will be phased into the remaining 10 senior high schools by September 1983.

The project goals are:

1) Reinforcement or review of information gained in the classroom for students to grasp subject matter that is perceived as difficult for them.
2) Development of problem solving skills using programming algorithms or microcomputer capabilities to increase student ability to solve multi-disciplinary problems.
3) Use of simulation software for laboratory investigations that would otherwise be too time-consuming, hazardous, or expensive

for high school investigation (e.g., long-term genetic investigations, ammonia production, electric charge on atomic particles).

4) Interfacing (connecting laboratory equipment with the microcomputer) for investigation and analysis of data previously not readily available to students (e.g., continuous pendulum-period analysis, rapid cooling-curve plotting, genetic statistical analysis).

Each science department of the 12 high schools received four Apple II microcomputers, four monitors, four disk drives, one printer, and miscellaneous accessories. Each school received about 30 software programs for immediate use. Other software was loaned from a central collection. Schools purchased evaluated and approved software to meet their local program needs.

Four 3-hour follow-up meetings of the summer participants were held during 1982-83 to exchange ideas and programs and describe diffusion and dissemination activities done at the school level.

Products of the project included:

1) A model 10-day in-service program for training teachers.
2) Design and implementation of several laboratory investigations using thermistors and a teacher-made interface module to connect the experiment to the microcomputer through the paddle port analog-to-digital converter.
3) Implementation of an evaluation form and selection process for approving software for school use, and a data base storage system of the information.
4) Data collection on microcomputer use during school and non-school hours.
5) Lesson and unit plans with descriptions of approved software coded to each science course by topic and lesson objective.
6) A bi-monthly newsletter to schools on new products, new applications, and school implementation activities.

Data on the results of the project showed an unusually high rate of diffusion of the training among other teachers in the schools, students, and even parents. There was a rapid increase in microcomputer use by students. Many microcomputer programs were written by students and teachers and shared among schools.

ABSTRACT: The Function Game: Using a Microcomputer to Improve Graphing Skills

Edward Zeidman, Division of Science and Mathematics, Essex Community College, Baltimore County, MD 21237

Computers are gaining widespread use as instructional tools in the classroom. They have shown promise in such traditional tasks as drill and practice, question and answer formats, and data management. These are important applications of the computer, but in order to realize the full instructional potential of this relatively new medium, we must look beyond the above applications to ones which were previously not possible and for which the computer is especially well suited.

The graphing of functions receives a great deal of attention in the community college mathematics courses. However, student performance in this area has been poor. This is particularly distressing in light of the importance of functions and graphing, both to applications and to further work in pure mathematics.

The microcomputer, with its unprecedented ability to do tedious calculations almost instantly, makes it a natural choice for material intended to improve students' understanding of functions and graphs. This paper explains a computer activity designed to provide students a fertile, mathematically accurate environment and the motivation to manipulate that environment to learn about graphs of equations. This activity is called the Function game.

The Function Game is a computer educational game designed to aid the student in learning to recognize functions of a single variable - a necessary skill for mathematical modelling. The Function Game challenges the student by effective use of computer graphics, immediate performance feedback, scoring, and competitive skill ratings. The student is able to explore and discover the association between graphs of functions and their equations, with the computer giving the student immediate feedback.

The student starts the game by selecting functions he/she wants to study. (The game package comes with a set of over seventy functions, and more can be added.) A function is randomly chosen from this set; the graph is drawn; the student then examines it. The student can imagine that he/she is a detective, whose job it is to solve the case of the "mystery" function. The student must guess the name of the function and determine the value of its unknown constants. Then the graph of the guess is drawn, overlaying it on the graph of the actual function. The student can visually ascertain the correctness of her/his answer. (This provides the student with immediate performance feedback.) In keeping with the game aspect, the student is scored on the proximity of the actual function to her/his guess.

An important feature is that hints are provided when the student wants help. A set of hints comes with the package, but as with functions, these can be modified.

A special editor is provided to modify and add to the list of functions and hints. The instructor can even design entirely different sets of functions and hints, to apply to a physics, chemistry, or economics course. The editor is designed so that the teacher can tailor the program to fit his/her needs without knowing anything about programming.

The Function Game has been class-tested in precalculus and calculus courses at Essex Community College, Baltimore County, Maryland. We found that the program worked most effectively with groups of three or four students. The interaction among the students was very important in their learning process. Instructors remarked on how quickly they were able to identify the different functions.

The program, written to work on the Apple II microcomputer, will be demonstrated.

ABSTRACT: Computer Chronicon Project

Melvin H. Wolf, Professor of Humanities and English, Pennsylvania State University, Middletown, PA 17057

In their endeavors to achieve a better understanding of the past as a step toward developing clearer views of the present and future, scholars and students in the humanities have long made use of two standard forms of chronological tables; the first provides chronological lists of items, and the second horizontal or vertical timelines. Both forms are designed to facilitate perceptions of relationships between various events and persons, but the utility of both forms - as they are now generally available - is diminished by their being set on printed pages by the choices of persons other than the users themselves.

With computer terminals now readily available to investigators in all fields, it is time for the development of a machine-readable data bank of chronological informantion which a user can query with an eye to his particular needs and interests. Just as library users can now broaden or limit the range of responses to their bibliographic questions at library computer terminals, they should be able to broaden or limit the range of responses to their chronological questions at similar library, university, or classroom terminals.

The purpose of this project is to develop a Computer Chronicon capable of generating customized chronologies to users' specifications. The specific objectives of the pilot project now in progress are to:

1) compile a sample machine-readable data bank of chronological information,
2) develop a working set of computer programs which will a) generate both cathode ray terminal (CRT) and hard-copy output b) of both list and timeline chronological tables c)using both data bank and user-supplied data d) for both interactive and batch processing, and
3) provide full and readable documentation describing the system and its use.

WHERE ARE WE GOING IN THE USE OF COMPUTERS IN PUBLIC EDUCATION

Sylvia Charp
Editor-in-Chief
Technical Horizons in Education

The use of computers in education covers a wide range of activities and applications and its use in schools has increased tremendously. We are currently being swept along with the tide, using buzz words such as computer awareness, computer literacy, computer based instruction, etc., but without sufficient thinking concerning the direction we wish to go, profiting from experiences of others and then developing a plan for implementation. There is no doubt that many aspects of our lives will be changed, adapted, or modified by computer technology and that education must prepare students to live in a technological world. However, "to get on the bandwagon" should not be the prevailing desire. We now have sufficient accumulated experience and research to help determine our direction, though what should be taught and to whom is still being debated. We do have alternatives, and we can determine our needs and wants and better evaluate our options.

Computers in the Undergraduate Mathematics Curriculum

Sheldon P. Gordon
Suffolk Community College

ABSTRACT

This session is designed to explore some of the most sophisticated and novel uses of the computer in the undergraduate mathematics curriculum.

At Brooklyn College, CUNY, a group of faculty have developed and explored a multi-faceted approach to the use of computer to perform the clerical and routine work involved in producing drills, worked examples and related instructional materials. The main direct and immediate impact on education at the College has been the establishment of a math workshop, staffed by faculty and student tutors, to provide a major component of the math remediation needed by many students. The workshop is used by approximately 1000 per year and is based on the use of drill materials generated by programs developed within the project, largely by Professor Kovacic.

Other major areas of development and exploration at Brooklyn College include; 1) syntax driven authoring languages and programs to support them; 2) the production of materials for computer literacy and computer science education that illustrate the workings of various algorithms by printing traces of their operations; 3) the use of computer graphics on large and small machines to produce instructional materials for distribution and use in classes; 4) the output of material designed this way to vary high quality typesetting devices for educational publishing.

At SUNY College at Brockport, a group of faculty has developed a number of instructional modules for use in lower division undergraduate mathematics courses. The modules are used for problem solving, tutorial and modeling in differential equations, calculus and precalculus mathematics courses.

The mathematics faculty are using many of these modules for class demonstrations, particularly the function graphing and modeling programs. Several modules are designed to be used as laboratory supplements to classroom lectures. These modules include suggested laboratory exercises. Students have access to the materials in a departmental laboratory equipped with several Apple computers networked to a hard disk.

At the University of Pennsylvania, the faculty has adopted the philosophy that computing can be used in mathematics instruction to motivate the mathematics and to enable the student to solidify the grasp of mathematical concepts. As an illustration of such usage, two versions of a course entitled Senior Seminar in Computational Mathematics will be described. This course is a required course for computer math majors at the University.

The first version of the course is An Introduction to Computer Simulation. This course involves mathematical modeling, differential equations and probability theory. The second version of the course is the Mathematical Foundations of Computer Graphics. This version requires proficiency in linear algebra and some geometry.

PARTICIPANTS:

Michael P. Barnett
Brooklyn College
City University of New York

Theron D. Rockhill
SUNY College at Brockport

Gerald Porter
University of Pennsylvania

Simulation: A K-College Teaching Strategy

Beverly Hunter
Targeted Learning Corporation
Amissville, VA 22002

SPONSOR: Society for Computer Simulation

ABSTRACT

Computer simulation is a widely used problem solving aid in many areas of life. A variety of simulations, from board games to role playing, have been traditionally used by teachers. With the increasing availability of computer resources, it is likely that computer simulations will become widely used in classrooms.

As with every pedagogical strategy, educators should be cognizant of its benefits and drawbacks. The purpose of this panel is to clarify some of the issues surrounding the use of computer simulation as a teaching/learning tool. Discussion topics will include:

1. a recapping of the history of computer simulation models as classroom aids;
2. an overview and assessment of commercial simulation models available today;
3. specific examples of integrating simulations into the curricula;
4. a look at the development processes of simulation models;
5. the inherent dangers of using canned simulations; i.e., the black box phenomenon;
6. alternatives to using canned simulations - or teaching students to be model builders.

The panel represents some of today's leaders in both the academic and commercial community of model and simulation advocates. The panelists experiences span all grade levels, from primary through college.

PANELISTS:

Alfred Bork
Education Technology Center
University of California, Irvine

Ludwig Braun
New York Institute of Technology
Old Westbury, NY 11568

Jonathan Choate
Groton School
Groton, MA 01450

Tom Snyder
Tom Snyder Productions, Inc.
Cambridge, MA 02138

Considering the Lack of Instruction Computing in Higher Education: Why?

Lincoln Fletcher, Moderator
MECC State University Instructional Coordinator
St. Paul, MN 55119

ABSTRACT

There is far less happening in the area of instructional computing - using computers as an instructional tool - in higher educational institutions than in elementary and secondary school across the country. Why is this the case?

This panel will discuss some of the issues and problems related to this question. Members will represent different perspectives on the situation: Computing Center Director, faculty member, administrator, and computer coordinator. Through discussion and sharing of concerns perhaps some new understanding of the problems and some possible solutions can be found.

PANELISTS

James W. Johnson
Director of Information Technology
Univer ty of Iowa
Iowa City, IA

George Culp
Assistant Director of Instructional Computing
University of Texas
Austin, TX.

Charles Parson
Assistant Professor, Geography Department
Bemidji State University
Bemidji, MN

SPONSOR: ACM SIGCUE

The Funding Game: Playing to Win

John T. Thompson
Barker Central School
Barker, NY 14012

ABSTRACT

This session describes the grantsmanship process inherent in applying for private monies for microcomputers. The audience will leave the workshop equipped with a broad background in applying for such funds with detailed references to monies available through the large computer companies (e.g., Apple, Radio Shack, Atari). Contact persons, telephone numbers and extended bibliographies in the grantsmanship area will be given. Fast-paced and packed full of useful insights into the grantsmanship process, this session will appeal to beginning grant writers, plus established money seekers looking for a quick update and new information.

Attention is first given to the private foundations which are depicted by type (e.g., independent, corporate, community) and area of funding priorities. Audience is informed of the regional collections of the Foundation Center in public libraries, specific books on private foundations and addresses to write to for more detailed information on their individual needs. Emphasis is placed upon how to establish initial contact and rapport with foundation trustees.

Attention then will focus upon the grant application procedures for several of the large computer companies that have corporate foundations. The majority of the workshop will be devoted to these specific procedures.

Throughout the presentation, handouts will be distributed to the audience. The method of presentation will be a lecture format with questions from the audience.

DESIGNING A PROGRAMMING COURSE FOR MBA STUDENTS

*David V. Cossey, Director
The Wharton Computer Center
The University of Pennsylvania - The Wharton School
Philadelphia, Pennsylvania 19104
(215) 898-6422

David Rossien
EXXON Corporation
New York, New York

This paper discusses the redesign of a non-credit programming course for MBA students. At Wharton this course is entitled "Problem Solving Using the Computer" (BA814), and all MBA students must either waive the course by credentials (based on prior computer experience), waive the course by exam, or pass the course as offered by the Wharton Computer Center. Wharton has offered such a course for several years, and recently the course was redesigned. The study described in this report and the redesign took place in 1980-81. The principal issues raised during the redesign period concerned the goals of such a course, whether the existing course met those goals, and if not, was it possible to design a course better suited to meet the goals.

INTRODUCTION

For several years, BA814 used the programming language APL. The course consisted of approximately 22 hours of lecture presented by members of the Wharton Computer Center staff (generally MBA candidates), and was offered four times each year: in August, during the Fall and Spring semesters, and during the January inter-semester period. The August class met for three weeks, the semester classes for seven weeks, and the January class for two weeks.

All MBA candidates must either pass or waive the course in order to graduate. During the past two years the waiver requirements have been relaxed, so that "substantial computer experience" is sufficient to receive a waiver by credentials. About one-third of the recent matriculants currently waive the course by credentials.

APL was chosen for two principal reasons:

1. It is an interactive language. BASIC, the only other interactive language available, was not well implemented on the DECsystem-10.

2. It is a very powerful language, and it was hoped that knowledge of APL would provide a useful tool that students would utilize throughout their time at Wharton.

COURSE GOALS

Faculty and administrators have described the primary purpose of the course as "facilitation of a better understanding of the nature of computers and their capabilities". The course instructors and the Wharton administration believe that such an understanding is important to a manager, especially given the rapid influx of computer technology into the business world. The consensus is that an active, hands-on approach to computer programming is essential for a foundation of computer operation and utilization. In the course of our research we spoke with faculty and administrators from numerous graduate business schools throughout the country. Most stated that developing an understanding of computers is important; their respective schools concurred by requiring students to complete a course similar to BA814.

A secondary, but noteworthy, goal of the BA814 course is to provide the student with a tool, the knowledge of a programming language, which may be used while at Wharton.

Finally, a tertiary but relatively less important goal is to provide a tool which might be useful to the student after graduation. While it is not expected that any Wharton MBA student will be doing a significant amount of programming, it is felt that with more and more businesses gaining access to computing power, the knowledge of a high-level programming language might be useful to graduating MBAs. The widespread purchase and use of personal computers for home and office use might, in the near future, make this a more important goal.

DESCRIPTION OF THE FORMER (APL) COURSE

The 22 hours of class time that comprised

BA814 were designed to instruct the student in the fundamentals of the APL programming language. A simplified outline, indicating the approximate number of class hours spent on each subject, follows:

(2) Introduction to Using APL on the DECsystem-10
(2) APL Data Structures
(6) Using APL as a Calculator
(4) Writing Simple Programs (Functions) in APL
(3) More Complex Programs in APL
(5) Manipulating Matrices in APL
(4) Formatting, Flowcharting, Discussion and Review

There were three 1-2 hour 'quizzes', 8 assignments, each of which required one to three hours of computer terminal use, and a final project that generally took 8 to 10 hours of terminal time and an equal amount of outside preparation.

To pass the course, a student had to accumulate a satisfactory average on the exams (generally about 70%), and complete the assignments and the final project. Since there was no penalty for dropping the course, most students did so when it appeared they were not passing. Therefore, the number of students who receive grades of NC was quite low, around 5-10 percent. However, the course was sometimes subject to a high attrition rate; in January this was sometimes as high as 40 percent, and ranged between 15-20 percent in August, and 20-40 percent during the semester.

Students did not generally find the course conceptually difficult. There is no question that most students were learning what we have asked them to learn, though their responses on course evaluation forms suggested they did so grudgingly.

PROBLEMS WITH THE FORMER COURSE

The critical problems with BA814 as it was taught are outlined below:

1. The course did not teach concepts which are basic to an understanding of the role of computers in the business environment. Concepts such as structured programming, files, records, databases, and text editors were among those not covered in the course.

2. Students reported the course unduly detailed and time-consuming, particularly in light of its non-credit status.

3. The skill(s) taught in the course (e.g. the APL language) were not used again by the vast majority of students while at Wharton.

4. Students often found themselves ill-equipped to work with the computer packages which they do use while at Wharton, such as SPSS, EMPIRE, IFPS and TSP. This was because they had only been taught concepts relevant to the APL language.

APL is fundamentally different from almost all other computer languages. Though it is true that those differences make APL a powerful tool, they also set it apart from languages such as BASIC, FORTRAN, PL/I, COBOL, Pascal and Ada. APL often makes the computer transparent to the user, hiding what the user should really see. In this way it may distort both the "nature of computers," and their capabilities.

There is without a doubt an APL "mind-set," a way of viewing problems in a different fashion than one would if trained in one of the other languages mentioned above. The fact that APL is not used extensively in the business community, nor is it the primary teaching language of any other business school in our survey, suggests that emphasizing the APL "mind-set" does not necessarily facilitate the understanding of computers and their usage in the business world.

Proponents of the APL language view it as the language of the future. However, computor scientists have been hearing this claim for ten years. A recent survey, taken of the readers of BYTE magazine (BYTE, October 1980), shows the "literacy" rate of APL to have dropped faster in the past two years than that of any other computer language. The rate, according to this survey, is now 22 percent. The literacy rate (i.e., percentage people who "know" the language) of Pascal, on the other hand, had increased by over 150 percent, and is now at 40 percent, up from 14 percent two years ago. This suggests that 10 years time should be sufficient to determine whether APL is the "wave of the future". Admittedly, more and more companies are using APL, just as more and more companies are using computers, but the widespread utilization of APL that had been envisioned appears not to have taken place.

Virtually every BA814 course evaluation form used to come back with a comment about "the heavy workload". With three exams, eight assignments, and a long final project, BA814 compressed the work of a semester-long course into a few weeks. A significant part of the problem stemmed from the inherent nature of the APL language. There are so many different operators in APL that complex expressions must be taught before the student can really use its full power. It became apparent that it was not possible to teach APL in a less detailed manner.

since we felt that we had already pared the language down to its smallest possible subset, while still demonstrating the power of the language. We felt that it was not possible to further reduce the amount of time necessary to teach APL.

Experience over the past several years suggests that while APL may have been a useful tool at some point in time, the average Wharton student was not using his or her knowledge of APL in other courses while at Wharton. This was to a large extent directly traceable to the advent of the hand-held financial calculator and the development of special-purpose high-level software packages.

When the computer is used in a Wharton course, it is usually via a pre-written package which requires only that the user know how to log in and initiate the program, and not any knowledge of a computer "language". The packages which are most frequently used are LINDO, IFPS, EMPIRE, SPSS, BMDP, Minitab and THE MANAGEMENT GAME.

In addition to the use of the computer in regularly-scheduled Wharton courses, many students use the computer to assist them in their Advanced Study Project (ASP) courses. In almost all cases it is a package, and not a general-purpose language, which is used. Many marketing students analyze their questionnaires with SPSS, finance students use TSP or IFPS, and ASPs have also made use of EMPIRE, BMDP, and IDA. In some cases, databases such as COMPUSTAT are accessed, and since APL can only read specially constructed databases, a consultant must write a program in FORTRAN or Pascal for the student.

Many students did not know what facilities (other than APL) were available on the Wharton DECSystem-10. BA814 did not attempt to teach the students what tools were available on the Wharton DECSystem-10, and because of time constraints, it was not possible to conveniently include this knowledge in BA814. The experience of one of the authors of this paper (Rossien) suggests that the tools which students want to use now are not those which mimic calculators, but rather packages such as those mentioned above and programs with access to databases, which calculators cannot provide.

EXPERIENCE OF OTHER BUSINESS SCHOOLS

Part of this study included conversations with administrators and faculty at fifteen graduate business schools. Of the fifteen surveyed, twelve offered some sort of required course in computer programming as part of their MBA curriculum. The three that do not have such a programming course, have a formal course which utilizes computer packages, such as those previously mentioned, in their "core" courses. This suggests that a majority of these business schools believe that a course in computer programming is relevant to the MBA degree.

Each school that requires a course utilizes a different format. In about half of the schools, the course combines Management Information Systems and computer programming into one course. These courses are always for credit, and typically last a full semester. Some schools, such as NYU and Columbia, had course structures similar to that offered at Wharton; non-credit short courses which teach only programming. Other schools give one-half or one-third credit for their courses. No course lasts less than two and one-half weeks, and most short courses last for six weeks.

By far the most common language taught is BASIC. Nine of the twelve schools with programming courses teach BASIC, one teaches PL/I, one FORTRAN, and one a combination of PL/I and APL.

For our study, we requested syllabi from each school, and received about half to examine. Our findings indicate that most courses cover the rudiments of BASIC programming. A few simple programs are assigned, and some courses have a mid-term or final.

As far as programming as a tool for a student's use while in graduate school, most professors admitted that it was not used very much. Stanford noted that it had an extensive BASIC library (much like Wharton's APL Library), and that several programs were used by students, particularly the plotting and regression routines, but that a knowledge of BASIC was not essential for this use. Some courses, such as that at Carnegie-Mellon, spent a lecture on canned packages such as SPSS and text processing programs such as SCRIBE. The instructors at Carnegie-Mellon said many MBA students used the text editing facilities to write their papers, as well as to generate cover letters and resumes for job searches.

SUMMARY

1. Learning a high level computer programming language (in contrast to a "package" such as SPSS or TSP), is quite useful, and perhaps essential, to facilitating a good understanding of computers and their capabilities.

2. The advantages of teaching the APL

programming language have not been realized. For many reasons, APL was not used as a tool by the vast majority of MBA students, before or after they graduate. Students found the previous course to be much more time-consuming and difficult than they felt the situation warranted. Yet for all this work, students did not have a good understanding of what computers can do and how one might expect computer technology to affect the business environment in the future.

3. APL instructors agreed that because of its power and breadth it takes more time to teach the fundamentals of the APL language than it would to teach many other higher level languages.

4. The concepts and principles behind APL, as well as the language itself, are not at all widely utilized in the business world. Concepts such as files, records, and program structure were not really covered in BA814, as they are not an important part of APL. These concepts are considered important and integral parts of EDP and should be included in the MBA curriculum.

5. Packages such as EMPIRE, IFPS, SPSS and TSP are not as readily understandable to persons knowing APL as to those knowing "FORTRAN-like" languages such as BASIC or Pascal.

6. Most of the business schools we contacted have a required course in computer programming as part of their MBA curriculum. Most schools teach the BASIC language in one of its many forms. Usually the BASIC that is used is an enhanced version of BASIC.

INITIAL RECOMMENDATIONS

It was clear that the goals of BA814 were not being met in an effective manner within the structure of the APL course. After studying the course outlines and syllabi of other business schools, and speaking with Wharton faculty, staff, and students, we concluded that it would be possible to design a course to better meet our goals. Our initial design of such a course consisted of three parts:

1. An introduction to the Wharton Computer Center Facility

2. Instruction in a language such as BASIC or Pascal

3. Instruction in a package of the student's choosing

It was envisioned that each of these parts would be viewed as an independent "mini-course", and would normally be taken in the above order.

The goal of the first course segment was threefold:

1. Give the computer novice an overview of the very basic concepts of computer systems. These include:

   -How to log on, log off, get help

   -What are files, and how they are used

   -What is the significance of terms such as CPU, memory, disk, terminal, batch, interactive, on-line, programming language and canned-package.

2. Give the students some idea of what tasks can be easily accomplished on the Wharton DECsystem-10, and generalizing from this, on any large computer system in the business world. This means giving a description of the packages, databanks, hardware, and libraries available on this system as well as presenting a summary of what is available in today's business world, and where trends in such fields as office automation, computer modeling, and database management are headed.

3. Teach the student how to use a simple text editor

The above material could be covered in four to five class hours.

We concluded that some form of high level programming language instruction facilitates a better understanding of computers and their capabilities. The following is a list of concepts that should probably be covered in the introductory programming module:

1. Variables, numeric and character
2. One and two dimensional arrays
3. Conditional statement execution (If-Then-Else)
4. Looping and iteration
5. Subroutines and functions
6. Input and output to the terminal and to

files
7. Construction of a "Conversational" program
8. Program modularity, block structured code, hierarchial problem solving

The language of instruction should be one which is comparatively easy to learn, and allows students to study the concepts suggested above without being constrained by artificialities inherent in the language. A study of language choices suggests that the possible languages are all "FORTRAN-like", but there are several alternative languages to choose from. We felt that the best choices would be BASIC, Pascal, or PL/I. PL/I was not implemented on a DECsystem-10. When we began our study, we were not aware of an enhanced version of BASIC for the DECsystem-10, so our initial choice was Pascal. In the course of discussions about the proposed redesign of the course, we discovered an enhanced version of BASIC, which contained advanced control structures (IF-THEN-ELSE, etc.), long variable names, expanded user-defined sub-programs, and many other advanced features. Upon investigating this version of BASIC (MaxBASIC from National Information Systems), we decided to use BASIC in the Programming Language module of our redesigned BA814.

Topics in this section include the following (the numbers in parentheses are the number of hours estimated): Introduction to programming (1.5), structure of a computer program (1.5), variables (1.0), expressions and assignment statements (1.5), input/output (0.5), conditional execution (1.5), iterative looping (0.5), flow charts (0.5), one-dimensional arrays (1.0), strings (0.5), two-dimensional arrays (1.5), subroutines and program modularity (1.5), reading from files (0.5) and 'Putting It All Together' (1.5).

In the third section of the course the student would choose which of a half dozen or so packages he/she wishes to learn. This section of the course would have certain prerequisites, since most of the packages assume the user understands the nature of the problems he/she wishes to solve. A partial list of the packages which would be offered includes: SPSS, TSP, IFPS, EMPIRE, IDA, Minitab, Runoff (text processing).

The course would have one individual final project which would require the student to use the package to solve a real-life problem. Experience at the Wharton Computer Center suggests that is is definitely possible for the student to become quite facile in any of the above packages after at most five class hours.

As originally proposed, the new course would have required slightly more class time than previously.

As described above, each segment of the course attempts to meet specific goals. The students would have learned the fundamentals of computer systems and thinking, as well as the capabilities of the Wharton Computer Center. They would have been presented with three major areas of computer technology in business: text editing and word processing, data processing via a high level language, and computer packages which can be used to solve business-oriented problems.

FINAL IMPLEMENTATION

After the initial recommendations were distributed and discussed, a revised course began to emerge that was a modified version of the initial plans. We had initially recommended the use of Pascal for the Programming Language module of the course. After we discovered and evaluated the MaxBASIC language from National Information Systems, we recommended its use as the language to teach.

We decided that having three separate modules would create administrative problems, but we thought that the content of the three modules should somehow be provided, perhaps by alternative means. To this end the first module (Introduction to the Computer Center) was combined with the second module (Introduction to a Programming Language) in a single course that would be required of all MBA students without previous experience with computer programming.

The content of the third module (Introduction to a Package) is provided in optional short courses offered by the Wharton Computer Center. Packages taught to meet the objective of the proposed third module include: SPSS, BMDP, IFPS, EMPIRE and TSP.

Officially, BA814 now consists of the first and second originally proposed modules in a single non-credit course. This course is offered four times a year: August, Fall, January and Spring. In August the course runs for 7-8 class days, during the Fall and Spring the course runs for about 7 weeks for 2-3 hours per week, and in January there are six 3-hour sessions (Morday-Saturday). The January session is run the week before classes begin for the Spring semester. Enrollments for this course during the first year were as follows:

August, 1981	300
Fall, 1981	50
January, 1982	60

Spring, 1982	30
August, 1982	200

The course has been received very well, and we have found much less resistance to the revised course than we did to the previous course. We attribute this to two primary reasons:

1. The use of BASIC rather than APL
2. A more modest workload for the course

Many students either are buying or expect to buy a personal computer system within the next five years. A recent questionnaire of Wharton MBA students indicates that 46% of the respondents expect to buy a personal system in the next five years, and 40% of the respondents indicated that they have used a personal computer system. The proliferation of personal computers has made the teaching of BASIC more 'palatable' and logical for many of the MBA students who take BA814.

As a result of this redesign of BA814, and to supplement BA814, and to assist MBA students, faculty members, and other interested parties in selecting a personal computer system, the Wharton Computer-Center-sponsored-a-Computer Fair in September, 1981. The fair brought twelve local vendors to in an exhibit area. There were as many different systems on display as vendors. There were also lectures on various topics, including 'Personal Computer Systems: What They Are and What to Buy'. The 1982 Fair had twenty vendors and 1,500 attendees. We feel that the Fair has become an important part of the education programs provided by the Computer Center, and although not an official part of BA814, the Fair is in many ways an extension to BA814.

A CURRICULUM FOR A MASTERS PROGRAM IN COMPUTERIZED MATERIALS MANAGEMENT

by Daniel G. Shimshak and Dean J. Saluti

Department of Management Sciences
University of Massachusetts/Boston

Abstract

In this paper, a curriculum in computerized materials management is presented. The curriculum is designed for a graduate program and developed as the result of the combined efforts of (1) Cambridge College, (2) American Production and Inventory Control Society, (3) Digital Equipment Corporation, (4) Bay State Skills Corporation, and (5) the Boston Academic Community. The program is comprised of six modules of training, each lasting three months. Courses are designed to be taught in the evening and weekends to allow students to maintain employment or to enter optional materials management internships.

Introduction

This paper presents a curriculum in computerized materials management which currently serves as a pilot graduate program in Cambridge, Massachusetts. Such a curriculum although designed for a graduate program, can be modified for an undergraduate Bachelor of Science curriculum. In addition, the actual courses within the curriculum linked in such a way as to address various training deficiencies can then be used in specific industrial applications.

The field of materials management is one of growing importance. Industries involved in the production of equipment, components and materials comprise a major element of the economy and their productivity strongly influences domestic living standards and competitive positions in international trade. The materials manager must be involved with three major functions--systems design, systems operation, and systems control (see Stevenson [1] and Monks [2]). These encompass a wide range of management activities related to production, inventory, purchasing, stores, distribution and quality.[3] The computer has had a significant impact on recent advances in the field. Computers are used not only for solving problems dealing with materials management, but also for generating reports, automating manufacturing processes, controlling operations and in designing products.[4] Now, more than ever, is the computer and materials management linked together.

From a historical perspective, the need for a masters degree program in computerized materials management evolved from two sources--minicomputer vendors and manufacturing practitioners. Digital Equipment Corporation (DEC), the largest American manufacturer of minicomputers, had recognized the fact that their PDP 11 family and the new VAX models supported a major proportion of materials management installations. DEC's field service representatives and sales personnel had come to the realization that the users at these installations had skill level inadequacies which prohibited proper system applications. For example, users with materials management training lacked computer skills while technical computer personnel failed to understand materials management techniques. Furthermore, the users had begun to speak out and sought any and all available training sources.

DEC's formal recognition of this problem was matched by APICS, the American Production and Inventory Control Society. APICS, a professional association for materials management practitioners, clearly voiced a need for education in computerized materials management. An informal task group of Boston APICS members. DEC representatives and University professors began to work together to develop a curriculum to meet the needs in the field. What evolved was a pilot program with support from several sources:

(1) Cambridge College--a small private college accredited to grant masters degrees in management, contributed institutional resources which will lead to a graduate program in computerized materials management.

(2) APICS--the Boston Chapter membership donated time and resources to the development of a curriculum which reflects specific skill levels recognized by practitioners in the field.

(3) DEC--contributed a PDP 11/34 minicomputer system to be housed at Cambridge College to support all program training. In addition DEC provided technical input to the curriculum development and made a heavy commitment to program delivery.

(4) Bay State Skills Corporation--processes grants for the development of high-technology oriented training through funds allocated by the Governor of Massachusetts and the State Legislature. They provided funding for this pilot program.

(5) Boston academic community--representing University professors throughout the state from academic disciplines such as computer sciences, management, management sciences, and management information systems. They played an integral part

in the development of this curriculum. These professors held positions at institutions such as the University of Massachusetts/Boston, Southeastern Massachusetts University, Suffolk University, Bentley College, Northeastern University and Boston University.

The current economic status of Massachusetts supports the demand for positions in computerized materials management.[5] Massachusetts has the second lowest unemployment rate of the industrialized states. With regard to manufacturing activity and employment, Massachusetts statistics are much healthier than most of the remainder of the nation. Also it is critical to note that the most important piece of tax limiting legislation in the state, Proposition 2 1/2 which was approved in November 1980, had had its major impact on nonmanufacturing employment. Thus economic indicators reveal a maintenance of employment rates and economic stability in Massachusetts which is essential for continued growth in this area of computerized materials management.

Program Objectives

The first objective of this program is to build a foundation of technical skills in materials management. These are primarily mathematical skills with manufacturing applications. For example, inventory control draws upon algebra and probability theory, quality control relies on statistical analysis, and shop floor scheduling applies matrix algebra. Whereas mathematical analysis of these problems has until recently been performed by hand, today reliance on the computer has become mandatory.

A second major objective is to develop the ability to write programs to perform materials management functions. Teaching emphasis is placed on the development of the students' ability to design systems in a structured logical manner in virtually any and all applications and systems languages. This is accomplished by providing an environment which will require students to build systems (such as Materials Requirements Planning --MRP) in the following applications languages: Basic, Fortran IV, Fortran 77, Watbol, Cobol, PL/1, and PL/C. The student completes the program with a portfolio of materials management programs, each materials management technique being programmed in as many as five different languages. In addition these same techniques will be programmed in PDP 11 Assembly in order that they may acquire systems programming expertise and familiarization with macros. Proper programming techniques are emphasized such as structured design and coding as well as systems documentation. In meeting this objective students will gain the ability to effectively link materials management skills and techniques to computer applications and programs.

The third objective is to develop in the student an expertise in minicomputer operations. It is important that the students can configure a minicomputer system within a manufacturing environment; can plan a conversion in hardware installation; and can actually operate and manage a minocomputer system. Although generic minicomputers operating system's functions are presented, emphasis is placed on the knowledge of the DEC PDP 11 due to its popularity and applicability in the manufacturing environment.

Finally the program attempts to prepare the student for the interpersonal dynamics of the manufacturing work environment. Students must learn to work in a project task group setting where technical computer and manufacturing tasks must be shared and accomplished by a team of professionals. Various interpersonal skills and management principles, essential to success in this area, are taught.

Curriculum Tracks

The program curriculum can be found in the Appendix. This program is designed to be taught in the evening and weekends to allow students to maintain employment or to enter optional materials management internships. There are six modules of training, each lasting three months: thus the student may acquire a masters degree in computerized materials management in one and a half years. The admissions criteria provide an opportunity for individuals seeking a career change as well as those who have a background in business or manufacturing. Various courses such as accounting techniques or statistics can be waived or required given the student's previous educational background.

Track I -- Materials Management Techniques--includes the following courses:

- Basic Production and Manufacturing Principles
- Bill of Materials Techniques
- Capacity Requirements Planning
- Purchasing
- Introduction to CAD/CAM (Computer Assisted Design/Computer Assisted Manufacturing)
- Introduction to Robotics
- Forecasting Techniques
- Shop Floor Control
- Inventory Control Systems
- Manufacturing Production Scheduling
- Basic Principles of MRP
- Advanced Principles of MRP.

This track provides a technical expertise in nearly all areas of materials management as endorsed by practitioners and professional associations [illegible] as APICS. The traditional educational environ[illegible] these courses would allow the student t[illegible] thematical techniques to applications. [illegible] program moves forward to provide the tools for the design of computer systems for the specific materials management techniques. The logical progression of the courseware begins with the basic materials management principles, moves through specific techniques, and concludes with the all-encompassing systems environment of MRP.

Track II--Applications Programming for Materials Management includes the following courses:

Introduction to Computer Programming Logic for Manufacturing (Basic)
Intermediate Computer Programming Logic for Manufacturing (Fortran IV)
Intermediate Computer Programming Logic for Manufacturing II (PL/1)
Advanced Computer Programming Logic for Manufacturing (Cobol)
Systems Programming Techniques (PDP 11 Assembly).

The intention of this track is to familiarize the student with various applications languages while integrating materials management programming assignments with programming techniques. The student will learn Fortran IV by writing Bill of Materials programs, for example. The number of languages taught will depend upon the availability of compilers for the PDP 11/34; thus languages such as Watbol, PL/C and Fortran 77 might also be introduced to the student. Macros and systems programming skills will be acquired through PDP 11 Assembly language.

Track III--Computer Systems Skills for Minicomputer Applications--includes the following courses:

Introduction to Computer Applications in Manufacturing
Computer Law
Computer MRP Systems I.
Computer MRP Systems II.

Within courses such as Introduction to Computer Applications in Manufacturing and Computer MRP Systems I and II students will become familiar with commands for various software packages that are available from vendors (Original Equipment Manufacturers or OEMs) for the DEC PDP 11. Computer Law is an important part of the curriculum given that practitioners must continually negotiate contracts with hardware and software vendors/OEMs who offer materials management processing resources. In addition DEC RSTS Operating Systems and DEC Systems Manager topics are included within the content of specific courses in this track.

Track IV--Mathematics Workshops--includes six optional, noncredit mathematics workshops and a Statistics Applications course. As quantitative methods are taught within each course in the Materials Management Techniques track, the student is offered support in a workshop environment. Thus students with an innate fear of mathematics can be assisted. A basic statistics course in included which can be waived by students who have achieved a grade of B or better in a college statistics course. Most materials management techniques incorporate statistics principles thus justifying the statistics requirement.

Track V--Manufacturing Business Environment --includes the following courses:

Introduction to Manufacturing for the Business Environment
Accounting Techniques for Materials Management
Financial Techniques for Materials Management.

Again the opportunity exists for students to waive these courses if they have achieved a B or better in similar college courses. The purpose of this track is to acquaint students with important principles of the business environment that they will undoubtedly encounter in the performance of their materials management functions.

Track VI--Behavioral Dynamics--includes Management Style for Manufacturing, Job Search Techniques and six human relations, skill building courses taught within a workshop environment in each module. The topics covered in the Human Relations courses include interpersonal communications. team building, group processes, motivational techniques, leadership, conflict intervention, organizational effectiveness and integration techniques. A course in management style is included because it is perceived that graduates of this program will quickly move into entry-level management positions. Finally the course dealing with job search techniques will assist graduates in moving into immediate employment.

Summary

A consortium of contributors from the manufacturing industry, computer industry and academia have worked together to develop a graduate program in computerized materials management. The intent of the contributors was to develop a program curriculum with the following primary objectives:
(1) to build a foundation of technical skills in materials management,
(2) to develop the ability to write programs to perform materials management,
(3) to develop an expertise in minicomputer operations, and
(4) to prepare for the interpersonal dynamics of the manufacturing work environment.

The courses are offered in six modules and assembled within the following tracks:
(1) Materials Management Techniques,
(2) Applications Programming for Materials Management,
(3) Computer Systems Skills for Minicomputer Applications,
(4) Mathematics Workshops,
(5) Manufacturing Business Environment, and
(6) Behavioral Dynamics.

The pilot class will begin in July 1983, with an expected enrollment of 30 students. Continuous input will be solicited from program developers and program graduates. In this way the courseware will remain within the state of the art.

Appendix

Computerized Materials Management Proposed Curriculum

Module I	Contact Hours
Introduction to Manufacturing for the Business Environment	20
Basic Production and Manufacturing Principles	40
Introduction to Computer Applications in Manufacturing	20
Introduction to Computer Programming Logic for Manufacturing	80
(Human Relations Course)	40

Math Workshop I (Optional)	40
	240

Module II

Accounting Techniques for Materials Management	40
(Human Relations Course)	40
Intermediate Computer Programming Logic for Manufacturing	40
Bill of Materials Techniques	40
Capacity Requirements Planning	40
Math Workshop II (Optional)	40
	240

Module III

Financial Techniques for Materials Management	40
Statistics Applications	40
Intermediate Computer Programming Logic for Manufacturing II	40
Purchasing	40
(Human Relations Course)	40
Math Workshop III (Optional)	40
	240

Module IV

Introduction to CAD/CAM	20
Introduction to Robotics	20
Forecasting Techniques	40
Advanced Computer Programming Logic for Manufacturing	40
Computer Law	20
Shop Floor Control	20
(Human Relations Course)	40
Math Workshop IV (Optional)	40
	240

Module V

Inventory Control Systems	40
Master Production Scheduling	20
Basic Principles of MRP	40
Systems Programming Techniques	40
(Human Relations Course)	40
Computer MRP Systems I	20
Math Workshop V (Optional)	40
	240

Module VI

Advanced Principles of MRP	40
Management Style for Manufacturing	40
Computer MRP Systems II	60
(Human Relations Course)	40
Job Search Techniques	20
Math Workshop VI (Optional)	40
	240

References

(1) William J. Stevenson, Production/Operations Management, Richard D. Irwin, Homewood, IL. 1982.

(2) Joseph G. Monks, Operations Management, McGraw Hill, New York, 1982.

(3) Jack N. Durben, "Materials Management Online System," Proceedings of the 24th Annual International Conference of APICS, Boston, MA. October 6-9, 1982, pp. 27-30.

(4) Donna Hussain and K.M. Hussain, Information Processing Systems for Management, Richard D. Irwin, Homewood, IL, 1981.

(5) Roger J. Deveau, Dean J. Saluti and Daniel G. Shimshak, "Economic Opportunity--An Economic Profile of Massachusetts" in The Outlook on Massachusetts, Commonwealth Books, Palisades, NJ. 1982.

Information Literacy Course: A Recommended Approach

Eileen M. Trauth

School of Management, Boston University
Boston, MA 02215

Abstract

Given the pervasiveness of computers in our society,much recent attention has been focused on the development of literacy courses to prepare students for this new era. The typical approach is a computer literacy course which introduces the students to some programming language. This paper presents an alternative course on information literacy. The goal of information literacy is to be able to respond to the demands of an information-intensive society.

Introduction

The arguments supporting the need for a literacy course in the computing area have been made and are widely accepted. What is not so widely accepted, however, is the form that such a course would take. The rationale for computer literacy stems primarily from the pervasiveness of information processing technology in our society. Most fields of study require interaction with the computer resource. Upon graduation, virtually all areas of employment will bring the graduate in contact with information processing technology.

The response to this need ranges from a service course in the computer science department to specialized courses within the various disciplines. Literacy in this regard is generally labeled "computer literacy" and aspires to make the students proficient in the manipulation of a computer via some high level languge. The underlying assumption, it seems, is that learning how to successfully manipulate the technology is the best preparation for an information-intensive society. This paper suggests another approach, one based upon the goal of information literacy rather than computer literacy. According to this view, what should be the focus of attention is the information itself. The computer is viewed as a tool - albeit the major tool - used in the processing of information. Thus, it is studied - but from the viewpoint of its relationship to the information. Information literacy, then, is a broader concept than computer literacy. The goal is not competence in manipulating the computer, as the latter term implies. Rather, the goal is the capability of working with information in whatever form it arrives and by whatever means it is processed - via computer, or otherwise.

Fundamental to this view is the notion that information is a phenomenon that has independent existence. Thus, one could study information in much the same way as one could study energy, or music, or basket weaving. The emphasis would be on the issues associated with the existence of the phenomenon. Some of these are technological, others are behavioral. Thus, the spectrum of possible study ranges from systems design and programming to human information processing to societal impacts of information. These areas of study are, however, shaped by an "information perspective." This perspective means considering the phenomenon to be distinct from the media used for storage, processing or transmission. Since the focus is on the phenomenon and not the medium, a broader outlook is possible. Thus, the emphasis for problem solving would be on the satisfaction of the information need rather than on the manipulation of technology. As such, attention can be given to appropriate tools for information processing and transfer-whether it be print, electronic, video, etc. An additional aspect of this perspective is that it allows for the behavioral component. Since the various media involved with information are seen as being only the tools, the focal point can become the people - those affected by these tools.

Such an approach to literacy will enable the student, in this author's view, to cope with the issues surrounding an information-intensive society that go beyond manipulation of the tools. The student would develop a methodology for "learning to learn." Additionally, the student would develop an orientation toward information as an explicit and valuable organizational resource. Finally, assuming that most people will be users of information rather than developers of its systems and technology, they will be able to develop their ability to articulate their information needs and communicate them either to another person or to a machine.

Recommended Undergraduate Literacy Course

There are certain assumptions underlying the recommendations that will follow. First, it is assumed that this is a required course for all students in the academic unit - the department, school, or college. Second, for the majority of the students, this literacy course might be the only computer-related course that they will take. Third, it is assumed that one of the goals of such a course would be its integration into the subject matter of the major discipline. That is, this course would become an integral part of the student's educational experience. This reflects the attitude that the study of information should not be separated from the setting in which it is to be used. A final assumpton is that the study of information is not synonomous with the study of the technology used to process it. The latter is an aspect of the former.

In keeping with the emphasis on information literacy this course should, among other things, familiarize the student with the information environment. That is, the student should become comfortable with the process of articulating the information requirements of a given situation. In addition, he/she should be able to identify and understand the role of information flows within the organization. For this reason the course should include discussion of notions associated with information as an independent entity. The origins of informaton should be examined relative to the situations in which the students would be working with information. The properties of this phenomenon should be examined as well. In order to this, it is necessary to understand the distinction between data and information. While this may seem fairly obvious, there appears to be much confusion over the terms as evidenced in students' perceptions about the topic. A recent discussion in an introductory course yielded such responses as:

> "Data is what you have when you use computers; information is what you have when people do the work."

or

> "You take information and put it into the computer and do things like statistics on it and then you get data as the output."

The viewpoint held for this proposed course is that data should be understood to be the raw material out of which information is created. It is, therefore, information in potential. Data, when processed, does not necessariy result in information. It only does so when the recipient of the data is capable of understanding it and is motivated to do so. Some of the distinctive properties of information that can then be noted are the following: Information has subjective existence, while data has objective existence. Information is not depleted with use. Information is intangible. The overall intent of the treatment of information as a distinct entity with distinct properties is to separate the information content from the media or technology by means of which it is conveyed or processed.

A second major topic that should be addressed is that of systems. Many people use the terminology of information systems without ever having stopped to think about the implications of the term. Recommendations regarding the treatment of the term "information" have just been presented. Recommendations about the term "system" follow.

Students should first be introduced to the notion of a system in general (and accordingly, systems thinking). They should then probe the interaction of systems and information. Part of doing so involves examination of systems used for the processing and communication of information. These include software packages, new telecommunications offerings and manual procedures. Another apsect of the study of systems involves the study of the organizational systems within which information flows. If an information system is to automate the flows of information within the organization, then it is encumbent upon those working with such systems to be able to understand the existing framework that holds the information.

An overriding orientation that should influence the preceding treatments is the particular disciplinary setting. It has already been argued that information only has real existence within a given context. For this reason it is crucial to the introductory understanding of information and its systems that different scenarios of information generation and processing be used. Business students will encounter a different type of information than that for library science students. Liberal Arts students interact with information in ways much different from those of science majors.

A third major topic should be the study of information processing. In an era when the limits of computer processing continue to be expanded, students who may never take another formal course in this area need to develop a view of the entire spectrum. This involves treatment of three major areas.

If it is accepted that information is a phenomenon whose existence depends upon the characteristics of the recipient then it is necessary for the students to have some understanding of human information processing.

This includes treatment of both the ways in which humans process information in general and the ways that the particular people involved in the student's discipline process and use information. A secondary benefit is that this understanding is helpful when learning about machine processing.

Since commercially available software packages are increasingly the norm, students should have exposure to the types that they would be likely to use. They will then not only have experience manipulating "real world" software, they will also have the opportunity in an introductory course, to examine examples of more sophisticated software than they are capable of writing themselves.

The final aspect of information processing is to experience developing programs themselves. This is often the easiest segment of such a course. The students usually feel that they are working with something concrete that has specific outcomes. Despite the fears that some might have upon entering, by the end of the course the majority of students are usually quite excited about working with computers and are pleased with the acquisition of a new and valued skill. The challenge when incorporating a programming segment into such a course, however, is to maintain the proper perspective. The goal of the programming segment is not one hundred percent proficiency in some high level language. Rather, it is to understand how to process data into information by means of a computer program. The emphasis is placed on learning the input-process-output sequence. The students learn how programs work in general through writing specific programs. The particular language used is treated as a vehicle for conveying such concepts. With thirty to forty percent of the coursework devoted to computer processing only a subset of the language can be taught. The students are told this. They are taught enough to be able to see the way in which computers process data. Suggested constructs would be: simple input and output, sequential file processing, calculations, transfer of control and looping. By writing programs in a given language the students should develop a generalized understanding of how computers and software function. If some attention has also been given to human information processing then the students are able to contrast their own mental operations with the operations of the computer. This understanding has proven to be valuable when students encounter difficulties with theirprograms. A typical example is the holistic or gestalt approach to problem solving taken by humans as opposed to the linear approach taken by the computer. Another is the contrast between the amount of ambiguity each can tolerate.

The final topic to be covered in such a course should bring together the previous three. Through problems and cases the students should examine the ways in which information, systems, and organizations interact. Again, such a treatment should be geared to the disciplinary setting. In a business curriculum the students would consider the information problems of industry and the available ways of solving them. A literacy course for a liberal arts curriculum might emphasize societal impacts of new technology, coping with the "information explosion," etc. One way that such topics can be woven into the fabric of the course is to have each student give a five minute presentation at the beginning of class meetings. The students are told to report on a recent journal or newspaper article. They typically focus on leading-edge applications, societal issues, and the new types of technology that are emerging. These talks often lead to stimulating discussions. In addition, the students are made to feel that a portion of the course is governed by their particular interests.

The Goals of an Information Literacy Course

A very important outcome of such a literacy course is the demystification of the technonology. Students with no particular interest in or inclination toward computers develop a sense of self confidence. Through successful experiences with machine processing they are reinforced that with some knowlege humans can be in control of the technology.

Closely related is the second intent, that the technology be placed in the proper perspective: secondary to an understanding of the need for and the uses of information. The course should convey a user-oriented perspective. Since the majority of the students will eventually be users, this outlook is appropriate. Because majors should develop a sensitivity to the people for whom they will be working, this perspective is fitting.

The third objective can be inferred from the comments just made. This course should include both majors and nonmajors. Both groups will benefit from exposure to the other. Having a range of capabilities and interests poses a challenge to the teacher. This author believes, however, that the benefits are worth the extra effort. This type of literacy course provides nonmajors with an exposure to the entire field. By having majors take such a course as their introduction to the field they receive an overview of the range of issues rather than in-depth exposure to a narrow area (which is usually the programming dimension).

The final objective of such a course is to provide the students with some tools for coping with the information-intensive society in which they will be working and living. By learning about information and how it is processed they will develop skills in "learning to learn." With increasing obsolescence of knowledge, this is perhaps the best that can be hoped for from an education.

Conclusion

This type of literacy course has been taught in a variety of institutions to diverse types of students. Student reaction is consistently positive. In many ways this type of course is more difficult than one which treats a narrower cut of the subject area. The body of knowledge is large and is constantly growing. Most textbooks are not oriented to such a course. But given the pervasiveness of information technology and the need for all students to learn how to cope with it, such a literacy course appears to be a reasonable response.

A System for the Automatic Grading of Programming Style

Patricia B. Van Verth and Anthony Ralston

Department of Computer Science
State University of New York at Buffalo
Amherst, New York 14226

Abstract

With current emphasis on programming methodology in introductory computer science courses, automatic grading systems which merely check whether programs produce correct answers have become obsolete. This paper describes a method for automatically grading student programs for style using a system which implements a mathematical model of program quality. The quality metrics obtained from this system relate to the choice and use of data structures and control structures, and to the division of the program into procedures. A database of programs and grades is being created in order to test the system and provide material for further research.

Introduction

Relief from the burden of grading student programs is high on the wish lists of all computer science instructors. Student programs are a necessary evil in introductory programming classes and other computer science courses in order for students to demonstrate in a practical manner their mastery of programming skills. However, it is possible that performing the grading task could be relegated to the computer itself. This in fact has been done in the past for evaluating whether programs run correctly or compile correctly. With the current emphasis on programming methodology in elementary level courses, these automatic grading systems have become obsolete by virtue of the deemphasis on correctness: not only are programs expected to work correctly, they must also be well-written, thus demonstrating good programming style.

Programming style in the context of this paper is taken to be a combination of program appearance and program quality. Program appearance is that component of style that includes the format and internal documentation of a program. Program quality refers to the way in which the implementation of the algorithm has been accomplished, i.e. the way in which data structures and control structures have been used in the program. Quality in this sense is taken to mean the antithesis of the complexity of control and data structures. The research described in this paper is concerned primarily with automating the evaluation of program quality in student programs, and, in particular, program quality as it appears in introductory Pascal programs.

Benefits of Grading Style Automatically

Automatic grading of student programs places the process of evaluation on an objective basis since the automation itself requires that standards for judging have been defined and implemented. Often in the case of human graders lack of objectivity occurs when a set of programs is graded by several different persons or even when processed by the same person. This usually happens due to the absence of clear guidelines for style, with graders relying on personal opinion or intuition. In the case of several graders, these opinions often vary, thus, producing inconsistencies in the grades. In the case of an individual grader it is difficult to maintain the same level of consistency in the presence of large numbers of programs and lack of clear-cut standards. Moreover, time pressure on graders results in evaluations of program quality and appearance which are often too cursory. With an automated system, the definition of measurements implemented in a computer program means that those measurements will be applied in a manner which is impartial, effective, and consistent. This in turn should instill confidence in students that the assessment of programs is meaningful and fair. It is our opinion that an automatic grading system with well-defined measurements of program quality can do as well as expert human graders who take considerable care in their grading, and will do better than almost all human graders in practice.

As well as achieving objectivity, an automatic grading system would considerably alleviate the shortage of personnel to conduct computer science courses and to do the grading associated with these courses. Grading projects is a lengthy, time-consuming and boring task. It is beneficial to an instructor when the instructor obtains feedback on the effectiveness of

instruction; however, most times the instructor does not do the grading and gets only a superficial impression of how students are performing in this aspect of the course. Indeed, data from an automatic grading scheme could give an instructor far better feedback than is normally achieved. On the other hand the student derives the maximum benefit if the grading has included constructive criticism and praise. Given the size of classes and the number of persons available to do the grading, this is an almost unreachable goal within reasonable time constraints. An automated system would be capable of providing similar analysis in greater depth and with more accuracy.

Background of Automatic Program Graders

Programs for grading student programs have been in existence almost since the time when computer programming courses were first taught. The earliest known such system was developed in 1958 by Jack Hollingsworth at RPI and used to grade assembly language programs[4]. This program evaluated programs for correctness of the answers. Later versions of program graders shared the early version's concern with correctness, included efficiency checks for time and memory usage, and were extended to grade programs in a variety of high level languages[1,3]. In the early 1970's, emphasis in computer programming courses shifted toward programming methodology. This shift resulted in a relative deemphasis on correctness in evaluating programs. Other factors such as readability, quality and clarity became equal in importance to program correctness. Since a correct program in this context is one which produces correct answers for a standard data set or student-supplied data set, correctness can be readily checked by computer. However, evaluating a program for its style is a much more difficult task by virtue of lack of either a generally accepted definition of style or standards by which to judge style.

Some attempts have been made to relate program style to resource usage or program appearance[8,9]. These attempts have not gained wide acceptance since it seems clear that style encompasses more than is implied by these terms. Criteria used in grading style in programs usually includes the appropriate choice and use of control structures and data structures, and the division of the program into procedures. These criteria remain sufficiently ambiguous (note the word "appropriate") but can be rigorously defined by embodying them in a mathematical model of programs. Then the quality of the use of data structures, control structures, and procedures in a program can be measured using that model. This permits the comparison of programs performing the same task on the basis of these measurements, ultimately allowing one to determine that one program is better than another if the measures have been suitably chosen and the measurements are consistent with this conclusion.

To support such conclusions the measures derived from the model should include more than just simple counts of structures since a count essentially reflects which structures have been selected. To gauge the use of these structures, the measurements should also incorporate the effect these structures have on the program itself. Thus, measurements of program quality should include both syntactic and semantic analysis. E.g., it is not only important to note that a WHILE loop has been used in a program but also to recognize the effect that loop structure has on the flow of control of the program.

An Approach to Program Quality/Complexity

The approach to grading program quality discussed here originated in program complexity studies done by Enrique Oviedo at SUNY/Buffalo[5,6]. In that research Oviedo proposed a mathematical model of programs and from that model derived a set of measurements for program complexity. Program complexity is an area of software metrics which seeks to measure in a program the degree of difficulty a person has in understanding a program in order to debug, modify or maintain the program. Its importance arises from the considerable amounts of time and money which are spent in performing these three tasks in large computer systems. To date there are no absolute standards of program complexity; in fact there is no generally accepted model or measurement of that attribute[2]. Nevertheless, we see in the work of Oviedo the potential for developing an accepted model of program quality which can be used in the context of automatic grading for programming style.

Description of Complexity Measures

In his work, Oviedo models programs using the control flow graph of the program. Measurements for the complexity of control flow are made by counting the edges on the graph which relate to the number of potential branches in a program. Data flow measurements are made by computing data flow equations over the same control flow graph; these are related to the data complexity in a program. These measurements combine syntactic analysis, a parse of the control and data structures in a program, with semantic analysis, the effect each structure has on the control flow and data flow of the program, to arrive at the complexity measures. While no absolute standards exist, measures of this kind permit comparisons between programs implementing the same task involving essentially the same algorithm, i.e. one can ascertain whether one program is less complex than another by comparing complexity measures. If the model is appropriate, the results will identify those programs which perform the programming task in a less complex manner than others. This then provides the basis for an automatic program grader for programming style. See Figure 1.

Programs implementing the complexity measures were used as part of early experiments conducted by Oviedo. Based on these early experiments and reported elsewhere at this conference[7], the

```
PROGRAM COUNT(INPUT,OUTPUT);
CONST BLANK = ' ';
VAR
  NCHAR, NWORD, NLINE : INTEGER;
  OLDCH,NEWCH : CHAR;
BEGIN
  NCHAR := 0;
  NWORD := 0;
  NLINE := 0;
  WHILE NOT EOF DO
    BEGIN
      OLDCH := BLANK;
      WHILE NOT EOLN DO
        BEGIN
          READ(NEWCH);
          NCHAR := NCHAR + 1;
          IF (OLDCH = BLANK) AND
             (NEWCH <> BLANK)
             THEN NWORD := NWORD + 1;
          OLDCH := NEWCH;
        END;
      NCHAR := NCHAR + 1;
      NLINE := NLINE + 1;
      READLN;
    END;
  WRITELN(' THE NUMBER OF WORDS IS ',NWORD);
  WRITELN(' THE NUMBER OF LINES IS ',NLINE);
  WRITELN(' THE NUMBER OF CHARACTERS IS ',NCHAR);
END.
```

Data Flow Complexity = 20
Control Flow Complexity = 14

Example of Good Program

```
PROGRAM TEXTCOUNT (INPUT,OUTPUT);
CONST B=' ';
VAR   NCHAR, NWORD, NLINE: INTEGER;
      POS:   BOOLEAN;
      CH:    CHAR;
BEGIN
  READ(CH);
  IF CH<>B
    THEN POS:=TRUE
    ELSE POS:=FALSE;
  IF TRUE
    THEN NWORD:=1
    ELSE NWORD:=0;
  NCHAR:=1;
  NLINE:=0;
  WHILE NOT EOF DO
    BEGIN
    READ(CH);
      IF EOLN
        THEN
          BEGIN
            NCHAR:=NCHAR+2;
            NLINE:=NLINE+1;
            READLN;
            CH := B;
          END
        ELSE IF (POS=FALSE) AND (CH=B)
          THEN NCHAR:=NCHAR+1
          ELSE IF (POS=FALSE) AND (CH<>B)
            THEN
              BEGIN
                NCHAR:=NCHAR+1;
                NWORD:=NWORD+1;
              END
            ELSE IF (POS=TRUE) AND (CH=B)
              THEN NCHAR:=NCHAR+1
              ELSE IF (POS=TRUE) AND (CH<>B)
                THEN NCHAR:=NCHAR+1;
      IF CH<>B
        THEN POS:=TRUE
        ELSE POS:=FALSE;
    END;
  WRITELN(' THE NUMBER OF WORDS IS ',NWORD);
  WRITELN(' THE NUMBER OF LINES IS ',NLINE);
  WRITELN(' THE NUMBER OF CHARACTERS IS ',NCHAR);
END.
```

Data Flow Complexity = 69
Control Flow Complexity = 34

Example of Poor Program

Figure 1. Example Programs

The programs included in this figure are programs which count the number of characters, words and lines in a text file. Characters mean all characters including the end-of-line mark. Words are strings of characters excluding blanks which are separated by blanks and end-of-lines. Lines are signified by end-of-line marks. The most difficult part of the program is determining the word count, i.e. detecting where one word ends and the next begins.

The good program has lower control flow complexity than the poor one since it uses two WHILE loops and one IF-THEN-ELSE statement, whereas the poor one has one WHILE loop and eight IF-THEN-ELSE statements. The data flow complexity is lower in the good program since there are ten assignments or variable definitions with ten references to those definitions vs. eighteen assignments and twenty references in the poor program. The large difference in data flow complexity (20 vs. 69) between the two programs arises from the effect assignments have on references in data flow complexity measurements. At each reference, all of the previous assignments which might affect the value of a variable at the point of reference are taken into account, producing a multiplicative effect.

measures obtained using the complexity model compared favorably with the results obtained when programs were graded for complexity by expert graders. These early results imply that, in practice, when program quality grades are assigned by human graders, the automated complexity measures will compare favorably with these. Thus, the automated results seem to be capable of discriminating quality programs as well as human graders. These results have encouraged the current work in expanding the model and in developing a comprehensive automatic grading system.

Assignment of Grades

Since no absolute standards for program quality exist, i.e. a quality measure of 50 has no meaning by itself, part of the process of using the grading programs will be to calibrate the grades for the particular programs at hand. An assumption that must be made with this system is that programs being compared are programs which implement essentially the same algorithm and are thus capable of being compared. One approach to calibrating the system is to provide a "perfect" program against which to compare student solutions. To ensure fairness, the "perfect" (i.e. instructor's) program could be awarded a 95% grade to give the students the opportunity to arrive at a better solution. Student grades for quality will be assigned using the model values relative to those of the instructor's program.

Testing of the System

The grading system described here is unique since it not only automates the process of grading program quality, but it also tests this process against human graders to determine system performance. The testing portion of this research has been started by collecting the programs from several sections of introductory computer science courses using the Pascal language. Along with the sources for the programs, grades assigned to the programs are also being collected for later testing of the system. These are being entered into a database constructed for the purpose of providing sets of programs and grades for a variety of experimental purposes.

In order to make grades more meaningful for experiments, the recorded grades are sub-divided into grades for program correctness, program quality, program readability, internal documentation, output format, and external documentation. Program correctness includes the compilation and running of the program; full marks are awarded only if the program produces correct answers. Program quality includes those aspects previously discussed, i.e. choice and use of data structures and control structures. Readability includes the formatting of the program. Internal documentation refers to the commenting within the program text. We have also developed a method for evaluating automatically program format and internal documentation at the textual level. This method assumes that students have a set of well-defined rules for indentation and commenting; the automatic grader then measures how well the rules have been applied. Output format reflects the manner in which the output is presented. Note that program correctness does not include the appearance of the output. External documentation includes any separate documentation required as part of the programming assignment such as a discussion of the stepwise development of the program. The goal of such categorization of grades is to make students more aware of the various aspects of producing "good" programs. Hopefully, results from human grading will be sufficiently accurate to permit comparisons with automated measures in the various categories.

Implementation of the System

Implementation of the automatic analysis portion of the system has been essentially completed using a front-end parser for the Pascal language to derive control flow graphs and sets of defined and referenced variables from source programs. These graphs and sets are then processed by a complexity-measuring program that counts edges on the control flow graphs and calculates data flow equations. Programs are processed on a procedure-by-procedure basis for both types of complexity measures and the total for each type is obtained by summing over all procedures. This yields two components in a program complexity vector which is subsequently used for comparing programs.

In addition to the programs for measuring complexity, implementation of the grading system has progressed in three other areas: the collection of programs, the collection of grades, the construction of a database. Programs are collected by requiring all students to use a special job control language (JCL) command when running versions of programs to be handed in for grading. This JCL is very easy to use; it is no more complex than the normal system calls to the compiler and the linker/loader. The JCL controls the compilation and execution of the program, and it collects a copy of the source and output of the program for entry into the database. The student receives a compilation listing, output and two grading forms attached to the listing by the JCL. The grading forms have a coded ID number which permits later correlation with the collected copy. When the program is graded by the grader, the second form is copied from the first and retained by the grader before returning the listing to the student. The grades on the form which has been kept are then entered into the database and correlated with the copied source programs. To ensure consistency in the database, i.e. programs copied match programs graded, any programs and grades which do not have matching IDs are rejected.

This implementation also filters out programs which are syntactically incorrect (don't compile). And it also discriminates between programs which terminate normally and those which halt on some

error condition. Copies are kept of programs which halt prematurely; however, part of the encoded ID contains a record of the premature halt.

The system does not have the capability to check program correctness automatically since the primary goal of the research has been to automate style grading. Correctness at this point is a relatively easy task for the graders since student programs are run on standard data sets.

Database Description

Programs, program results and grades divided into categories are entered into the database. Also included in the database are specifications for the programming tasks and data sets. Programs can be accessed by task, section, student, semester. Thus far (Fall semester 1982) two projects of approximately 200 students each have been collected for entry into the database. Future entries include results from automated measures including those for measuring program quality. See Figure 2.

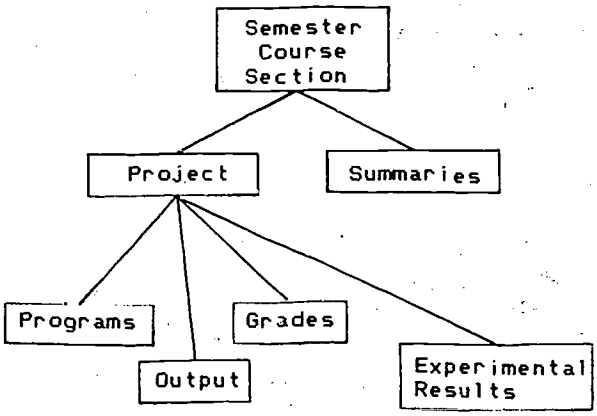


Figure 2. Database Model

It is anticipated that copies of the database will be made available to outside experimenters. We envision that this database can be used for further studies of the program complexity model, testing and comparison of other complexity models, and the establishment of a common set of programs to allow the reproduction of previous experiments. Having many copies of programs implementing the same task should provide the quantities of programs needed for experimentation. It seems obvious that an academic environment is the optimal source of these programs since industry could not tolerate the expense or justify the existence of so many copies of similar programs. Academic settings also permit samples to be collected from a variety of applications and programming languages. Many of these programs are considered "toy" programs by virtue of their size (100 - 500 lines, generally speaking). Nevertheless research must begin somewhere, preferably with large sample sets of programs. Most "real" world samples are available on a one-time only basis as implementations of very large systems.

Applications

Applications for grading systems are not simply limited to the evaluation of the end-product of student efforts. However, until absolute standards of quality are developed, we must be satisfied with this result. Once absolute standards are formulated, the evaluation process can be performed on individual programs, allowing students to submit programs directly to the automatic grader and obtaining immediate grades. A further refinement would incorporate the assessment into program development systems as part of an interactive set of tools used to write programs. In that setting the measurements could be used to direct students, or even professional programmers, to correct poor code, thus enforcing good programming habits at the outset. A not entirely unexpected side-effect of the automatic evaluation of program quality has been to produce an effective cheating-checker - it is not difficult to see that programs with identical complexity measurements are in fact likely to be the same program.

Conclusions

In this paper we have described an automated system for grading style in student programs. The system is based on a mathematical model of programs from which measures have been defined to measure the quality of a program. These measures of quality include the use and choice of data structures and control structures in a program; they permit the comparison of similar programs to determine which programs are better from a quality or style point of view. Preliminary testing of the system indicates that it performs in a manner comparable to human graders. It is expected that, after further extensive testing, the results will demonstrate that the model derived from the program complexity model is acceptable for modelling quality or style in a program and is a suitable basis for a system for the automatic grading of programming style.

References

1. Forsythe, G. and Wirth, N., "Automatic Grading Programs", CACM, May, 1965, pp. 275-278.

2. Harrison, W. et al., "Applying Software Complexity Metrics to Program Maintenance", Computer, Sept. 1982, pp. 65-79.

3. Hext, J. and Winings, J., "An Automatic Grading Scheme for Simple Programming Exercises", CACM, May, 1969, pp. 272-275.

4. Hollingsworth, J. "Automatic Graders for Programming Classes", CACM, October, 1960, pp. 528-529.

5. Oviedo, E., "Control Flow, Data Flow and Program Complexity", COMPSAC, December, 1980, pp. 146-152.

6. Oviedo, E., "Control Flow, Data Flow and Program Complexity", Dissertation, SUNY/Buffalo (to appear 1983).

7. Oviedo, E. and Ralston, A., "An Environment to Develop and Validate Program Complexity Measures", NECC, June, 1983.

8. Rees, M., "Automatic Assessment Aids for Pascal Programs", SIGPLAN, October, 1982, pp. 33-42.

9. Robinson, S. and Torsun, I., "The Automatic Measurement of the Relative Merits of Student Programs", SIGPLAN, April, 1977, pp. 80-93.

TEACH TOP-DOWN PROGRAMMING WHILE YOU TEACH BASIC

by Michael J. Streibel, Ph.D.

Instructional Systems Program
The Pennsylvania State University
University Park, Pennsylvania

Abstract

This paper describes a way to teach top-down programming principles while teaching the Basic computer language. Such an approach is especially important since microcomputers are appearing in large numbers in public schools and in homes and since Basic is the first computer language which most people encounter. Programs, in this approach, are designed by first spelling out the program goal as remarks in the header part of the program and then developing a complete set of major subroutine calls for the main part of the program. The subroutine calls in the main part of the program also include remark statements which spell out the goal of the subroutine as well as the parameters (if any) which are "passed to" and "returned from" the subroutine. Once this phase of the program-design process is finished, the actual subroutines are coded and tested to completion.

Introduction

Many computer literacy courses have been proposed in educational computing magazines which include a component on Basic programming. The approach taken in these courses invariable deals with teaching the individual statements of Basic before going on to higher-level programming concepts. At the same time, other articles in these magazines encourage their readers to use a top-down approach to designing programs. Why not combine the two approaches into one method and help your students learn the basic statements of Basic in the context of top-down-structured projects? What follows is a rationale and an example of this approach.

Top-down vs. Bottom-up Program Design

Anyone who has ever written a program in Basic has been tempted to turn to the keyboard as soon as possible and start coding the final program. The interactive nature of most interpretive Basics and the "friendliness" of microcomputers just makes this temptation irresistable. These same people have probably also learned the hard way that coding first and thinking later has led to innumerable difficulties - not the least of which is a premature commitment to specific solutions and a program of Basic code that eventually becomes unmanageable, unmodifiable, and ununderstandable.

The approach described above can at best be called the bottom-up method of designing programs (i.e., coding the individual statements of Basic before planning the overall structure of the program). It is only defensible when the programs are short (i.e., one CRT screen long).[1] In response to the bottom-up approach, several authors over the years have proposed a top-down approach to designing and writing programs.[2,3,4] Their argument goes something like this: the bottom-up approach does not teach effective problem-solving skills and usually results in poorly-written programs. Some people find the latter an acceptable state of affairs. Why, you therefore ask, should one learn effective problem-solving skills and why isn't a program that executes correctly enough? The answers to these questions are simple: programming is a subset of problem-solving (i.e., problem-definition and solution-generation techniques) and programs communicate with humans as well as with machines.[5] More of this later.

The top-down approach to programming attempts to define the problem clearly (i.e., the "top" if one views the process as a pyramid of levels) before breaking it into smaller and logically-distinct components. Giving a program a meaningful name, for example, is an excellent first (or "top") step. Issues on the most general level can then be articulated so that the eventual program has coherence and conceptual integrity (i.e., all the parts work in unison to achieve the program goal) and robustness (i.e., the program operates under many conditions).[6]

The "top", or first step of the design process, therefore involves giving a program a name and a description of what it will do. We usually do not think of this as programming but this step is as much a part of programming as writing specific Basic statements. We are, in effect, communicating with ourselves and others at this point (if we are part of a collaborative team) and thereby programming our minds to proceed with the problem-definition and problem-

solution in certain ways. The computing power of microcomputers is an extension of the human mind even though it resides in a separate box and has its own arcane rules of grammer and syntax. By describing what we are going to do in the program before working out how we are going to do it, we avoid locking ourselves into premature solutions. We will use REMark statements such as those in the program-example below to carry out the "top" of the design process. These will be described in greater detail later.

We proceed in the top-down design process by asking: what major steps will we have to take in order to accomplish the goal of the problem. Notice, we are leaving the how (or the specific code, whether it be Basic or Fortran or Pascal) until later. We are also attempting to make the major steps represent intelligent sub-processes of the larger goal rather than arbitrary divisions dictated by the Basic code. Hence, the major steps at this point are usually described as some sub-process like "define the values," "calculate other values," and "printout those values." The goal is not to get bogged down in the details of coding Basic statements when we are still brainstorming various ways to formulate possible solutions. Otherwise, we would be assembling the "bricks" of the program before designing the plan of the entire program. In the example that follows, a series of GOSUB and REMark statements will be used to carry out the second-level of the top-down design process. Each subroutine in the eventual program is treated at this stage as a black box that will do something for us. We can satisfy the urge to type something on the keyboard here by typing the GOSUB and REMark statements. If we want to change out minds about a particular solution, then it will be easier to delete a few lines of REMark statements in the Basic program than to delete a series of Basic code that we have worked long and hard to create.

The last (?) stage of the top-down design process is to fill in the actual subroutines and make them do what we want them to do. The large problem has been broken down into smaller and more manageable sub-problems that can be solved, coded, and tested by themselves. The trick is to treat each subroutine as a black box into which we pass some variables and values (i.e., input) and out of which we get some action, variables, or values (i.e., output). Since we have specified exactly what we want the black box to do, we can test and debug the Basic code in the black box until it satisfies our desires. The result is that, once we are finished with each subroutine, we can file it and forget it because we have solved a small part of our problem. In a larger sense, we have formulated the original problem in such a way that the power of computing can help us solve the problem. Notice that the last step in the design process involved coding the problem into a particular dialect of Basic. We could just as easily have coded it into another language because the design process up to this point would have been the same.

A Specific Example

What happens when you are teaching a computer literacy class and have to teach the individual statements of Basic? Do you throw out the top-down approach until after your students have learned Basic? Definitely not! Programming habits are developed and reinforced during every encounter with a computer. You can therefore structure a project in such a way that it is formulated and partially solved in a top-down manner. The example below will indicate how this is done (see Listing 1):

The object of the DATA.BASE program is to teach students about double-subscripted variables and data-structures (i.e., DB$(I,J) in line 3080 is a mailing list data structure with two subscripts I and J). Students are given the listing shown (in printed form and onto their disks) and then asked to complete the program so that it will carry out the goal described in the header. The main program is a series of GOSUB and REMark statements that has broken the overall goal into smaller sub-problems. Each subroutine is made up of a physically-distinct set of REMark statements and a RETURN statement.

The "top" of the design is the program name because it identifies the intent of the eventual program (see line 1020). Use meaningful words here so that you can refer to the entire program as a meaningful entity. An informative name is important because it communicates the intent of the program to other human beings. In many cases, that human being may be yourself if you develop the program over an extended period of time.

The program description elaborates the intent of the program (see lines 1060-1110). A clear description of the program in the early stages of the design process usually helps focus your attention on what you want to do without getting you bogged down in details. Many people object to this stage of the design process because they feel it limits their creativity and inventiveness later on. They have a point if the description is too detailed and prescriptive. You can avoid this pitfall be describing what will be done rather than how it will be done. For example, a bad program description would include specifications for the exact messages to be typed out to the user. In the DATA.BASE example in Listing 1, on the other hand, the description in the header does not specify how the program will carry out the intent of the author. The description is created by the teacher to help focus the student's attention on the goal of the program. The description is also written to serve as a model of how to formulate a problem.

A final component of the "top" of the program deals with a description of the variables and files that are used in the program and a listing of any formulae used in the program (there are no formulae in the DATA.BASE program). This constitutes a kind of glossary of terms so that teachers and students have a common "vocabulary"

```
1000  REM ------------------------------------------------------
1010  REM
1020  REM  PROGRAM: DATA.BASE
1030  REM
1040  REM  AUTHOR:  MICHAEL J. STREIBEL, PH.D.
1050  REM
1060  REM  DESCRIPTION: THIS PROGRAM CREATES A DATA BASE
1070  REM                IN MEMORY CALLED DB$ AND THEN ASKS
1080  REM                THE USER FOR A LAST NAME TO SEARCH DB$.
1090  REM                IF FOUND, THE CONTENTS OF THAT PERSON'S
1100  REM                DATA IS TYPED OUT, ELSE AN ERROR MESSAGE
1110  REM                IS PRINTED.
1120  REM
1130  REM  VARIABLES:
1140  REM            NE = # OF ENTRIES IN DATA BASE
1150  REM            NI = # OF ITEMS/ENTRY
1160  REM            MO$ = "YES" OR "Y" FOR MORE SEARCHES?
1170  REM
1180  REM            FOR I = 1 TO NE
1190  REM               DB$(I,1) = LAST NAME
1200  REM               DB$(I,2) = FIRST NAME
1210  REM               DB$(I,3) = STREET
1220  REM               DB$(I,4) = CITY
1230  REM               DB$(I,5) = STATE
1240  REM               DB$(I,6) = ZIP
1250  REM
2000  REM ------------------------------------------------------
2010  REM
2020  REM  MAIN PROGRAM
2030  REM
2040  GOSUB 3040: REM  CREATE DATA BASE DB$(1-NE,1-NI)
2050  GOSUB 4030: REM  ASK FOR LAST NAME; RETURN NA$
2060  GOSUB 5030: REM  SEARCH DB$(1-NE,1) FOR NA$; RETURN POINTER PT
2070  GOSUB 6030: REM  PRETTY-PRINT TITLES & DB$(PT,1-NI)
2080  GOSUB 7030: REM  ASK IF MORE; RETURN MO$="YES" OR "Y"
2090  REM
2100  IF MO$ = "YES" OR MO$ = "Y" THEN 2050
2110  END
2120  REM
3000  REM ------------------------------------------------------
3010  REM    CREATE DATA BASE DB$(1-NE,1-NI)
3020  REM    NOTE: TYPE IN YOUR OWN DATA STATEMENTS
3030  REM
3040  READ NE,NI: REM    NE=# ENTRIES, NI=# ITEMS/ENTRY
3050  REM
3060  FOR I = 1 TO NE
3070  FOR J = 1 TO NI
3080  READ DB$(I,J)
3090  NEXT J
3100  NEXT I
3110  RETURN
```

```
3120  REM
3130  DATA    3,6: REM              NE,NI
3140  REM
3150  DATA  "LAST","FIRST","STREET","CITY","STATE","ZIP"
3160  REM
3170  DATA  "LAST","FIRST","STREET","CITY","STATE","ZIP"
3180  REM
3190  DATA  "LAST","FIRST","STREET","CITY","STATE","ZIP"
3200  REM
3210  DATA  "LAST","FIRST","STREET","CITY","STATE","ZIP"
3220  REM
3230  DATA  "LAST","FIRST","STREET","CITY","STATE","ZIP"
3240  REM
4000  REM ------------------------------------------------
4010  REM  ASK FOR LAST NAME; RETURN NA$ TO BE SEARCHED IN DB$
4020  REM
4030  RETURN
5000  REM ------------------------------------------------
5010  REM  SERACH FOR NA$ IN DB$; RETURN PT=SUBSCIPT IN DB$ OR =0
5020  REM
5030  RETURN
6000  REM ------------------------------------------------
6010  REM  PRINTOUT DB$(PT,1-NI) OR ERROR MESG IF PT=0
6020  REM
6030  RETURN
7000  REM ------------------------------------------------
7010  REM  ASK IF MORE; RETURN MO$="YES" OR "Y" OR "NO" OR "N"
7020  REM
7030  RETURN
```

LISTING 1.

for the particular project (see lines 1140-1240). This mechanism serves a communication function. It also serves as an aid in directing the student's attention to the major data elements in the particular problem. Being able to construct a visual/spatial representation of data-structures helps one to think about data-structures.

We have now finished the "top" of the design process. We are fairly clear about what we want our program to accomplish, about the types of data objects (a mailing list in our example), and the specific variables to be used in our program. We are therefore ready to proceed to the next level of the design process.

The second level of the top-down design process is represented in the DATA.BASE program by the "MAIN PROGRAM" set of GOSUB and REMark statements (see lines 2040-2110). In this example, the teacher has already partitioned the problem in a certain way so that students are primarily concerned with the object of the lesson (i.e., double-subscripted variables and data structures). Let's take a look at the two main elements of this level of thinking.

The GOSUB statement is the only mechanism in most Basics which allows for multiple lines to be defined as a single unit. Why is this important? Well, we have already discussed the need to break a large problem into more manageable sub-problems and the value of leaving details until later. This helps us avoid premature foreclosure (i.e., premature commitment to specific solutions). Being able to cluster several lines of code together is also important because it allows us to have a physical counterpart to the conceptual units of the problem-solution. Humans can only deal with complexity at a certain level before having to "chunk" the problem into smaller meaningful units. Having these chunks in a physical as well as conceptual units is a definite advantage.[7]

The REMark statements in the "MAIN PROGRAM" are as important in the top-down design process as the GOSUB statements because they allow us to articulate our intentions for each sub-problem (see lines 2040-2080). Each subroutine, in effect, becomes the "top" of its own pyramid of code. Time spent in writing these REMark statements is not wasted if one is trying to brainstorm solutions to a complex problem because the REMark statements let us publically inspect our intentions and meanings. This allows us to "debug" our solutions on the semantic level before debugging the actual code. For example, we can ask ourselves whether we should ask for the users last name and immediately search the data base or whether we should do these things separately (see lines 2050-2060).

The REMark statements associated with each GOSUB statement also include a brief reference to what variables are passed to the subroutine and what variables are returned. For example, line 2050 indicates that we should jump to a "black box" of code in our program (i.e., line 4030) which will ask the user of the program for a last name and which will then return this name in a variable called NA$. The variable NA$ is not "really" returned in the program because it constitutes a global variable in Basic (i.e., can be used anywhere in the program), but we are treating it as if it were returned to us by the subroutine. This way of looking at subroutines fosters a computer-as-tool (i.e., subroutine-as-tool) attitude which is very healthy. Subroutines are treated as autonomous units of code that make no assumptions about the rest of the program (such assumptions are always a source of "bugs") except for what arguments are passed to it. The subroutine in the example initially contains only a RETURN statement (see line 4030) which students will then expand into a working subroutine.

The "last" level of the DATA.BASE program is made up of the individual subroutines (see lines 4030, 5030, and 7030). Since the purpose of our program is to learn about double-subscripted variables, the data-base variable DB$ is predefined by the teacher (see lines 3040-3230). This provides an example of how a data-structure is created in Basic and yet leaves open several activities for manipulating the data base. In the DATA.BASE example, students are asked to make up a list of names and addresses and then type them into the appropriate DATA statements.

The individual subroutines are made up of REMark statements which visually separate the code from the rest of the program and which describe what the subroutine is to do.[8] This may seem like a duplication of earlier efforts but it helps students focus their attention on each sub-problem of the program. The subroutine can then be coded and tested by itself since its input, output, and intent are clearly spelled out. The subroutine also gives the student wide lattitude in working out how the sub-problem is to be solved.

Summary

The top-down project approach to learning Basic which is described above has a number of distinct advantages over traditional instruction in Basic. First, it gives the teacher wide lattitude in formulating both the content and the structure of student projects. Hence, the teacher could fill some subroutines completely for beginning students and yet add extra options for advanced students.

Second, the approach described above lets the teacher address several levels of learning at the same time. For example, the project described above teaches about specific Basic statements (i.e., double-subscripted variables), computer concepts (i.e., simple data-structures), systematic program design, and how to think about problems. The assumption throughout the example is that computer literacy is ultimately a matter of the user's ability to think and communicate clearly (with humans and computers) about a

problem. Computing, in effect, becomes a well-defined extension of the human mind. The power of computing, however, is only engaged after the human being has clearly visualized and then articulated the purpose and parameters of a program.

Finally, the approach described above permits a problem to be solved with style and logic from the "inside out" (i.e., from an understanding of the problem).[9] Programs no longer look like "spaghetti-code" but reflect clear, efficient, and effective thinking. As a teacher gradually removes his or her own way of structuring problem-solutions, students learn to develop their own style of formulating and solving problems with the computer.

References

[1] Finkel, L. and Brown, J.R. APPLE BASIC: Data File Programming. New York: John Wiley & Sons, 1982.

[2] Dijkstra, E.W. A Discipline of Programming. Englewood Cliffs, N.J.: Prentice-Hall, 1976.

[3] Wirth, N. Systematic Programming: An Introduction. Englewood Cliffs, N.J., Prentice-Hall, 1976.

[4] Nagin, P. and Ledgard, H.F. BASIC With Style: Programming Proverbs. Rochelle Park, N.J.: Hayden Book Co., 1978.

[5] Lewis, W.E. Problem-Solving Principles for BASIC Programmers. Rochelle Park, N.J.:Nayden Book Co., 1981.

[6] Nagin and Ledgard, op. cit.

[7] Miller, G.A. "The magic number seven, plus or minus two: Some limits on our capacity for processing information." Psychological Review 1956, 63, 81-97.

[8] Nagin and Ledgard, op. cit.

[9] Dreyfuss, H. Designing for People. New York: Simon & Shuster, 1955.

USING COMPUTER SIMULATED MODELS TO TEACH PROGRAMMING LANGUAGES

BOGDAN CZEJDO

UNIVERSITY OF HOUSTON
WARSAW TECHNICAL UNIVERSITY

ABSTRACT

In this paper computer generated models to teach programming languages are defined and examined. The memory modeling, the program generation and the flowchart creation are presented. Generation and transformation of pictorial structures for each model is analyzed. The application of these models to computer-assisted instruction is shown. Finally further research in the area is suggested.

I. INTRODUCTION

The concept of model-based instruction was introduced in the paper "Using Model Based Instruction to Teach Pascal". On the basis of the principles of model-based instruction, CAI lessons have been designed for teaching elements of the programming languages. The early version of our system[1] displayed prestored course material, accepted only a restricted format of the answers and used built-in comments. This paper describes the research leading to the new version of the system. We took advantage of studies in artificial intelligence in the area of knowledge representation, program transformation, algebraic manipulation and system simulation[5-7]. Several simulation models were defined and implemented in Pascal on the VAX computer. They constitute the simulation system shown in Figure 1.

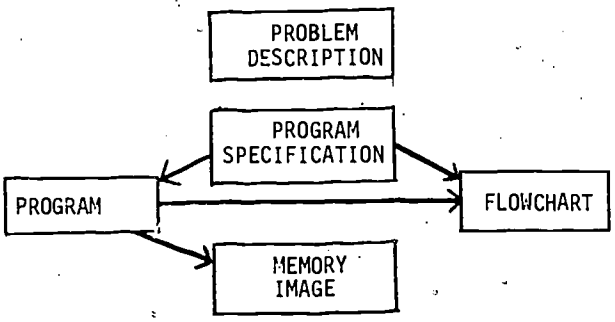


Figure 1. The Simulation System.

The program specification is the basis to generate the program or the flowchart. The flowchart may also be obtained directly from the program. Memory model is generated from the program. In Figure 1 there is no link between problem description and program specification because the current system cannot perform this transformation. We assume that the instructor will provide that link by means of associating each problem description with the corresponding program specification.

These models were chosen after a wide variety of structures had been examined and evaluated for potential use in the introductory courses in programming languages. For each model we investigated the problem of how to generate the solution (for the given task) and evaluate the correctness of the student's answer.

2. THE MEMORY MODEL

A good simulation model to start with in the introductory course is a structure which consists of an input device, memory, and an output device (external storage can be added later). This corresponds to the approach chosen in various textbooks for programming languages [4]. This model enables the student to visualize the execution of a single instruction as well as the whole program. Two phases of the operation of the model can be identified:

1. Creation of the structure
2. Transformation of the model

For the sake of generality, standard declarations in PASCAL and FORTRAN were chosen to build the structure. For example the following declarations in PASCAL:

```
I: integer;
Z: real;
Ch: char;
A: array [1..3] of integer
```

would result in the picture shown in Figure 2.

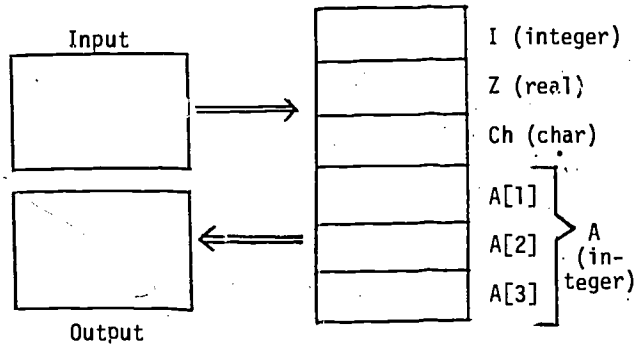


Figure 2. Generated Memory Model.

The generated picture consists of:

a) An Input Box
b) An Output Box

c) Memory divided into several boxes.

Several data items can be put into the input or output box. Mixing of types is allowed. The memory consists of several boxes in accordance with the declarations. Every box corresponds to a variable. In this way, a simple "visualization" of variables or constants in programming languages can be introduced. Big arrows indicate the allowed transfers.

Program instructions and input values are necessary to perform transformations of the model. The four permitted program instructions are:

a) read
b) write
c) assignment
d) new

For example, consider the Pascal instruction read-ln(I) and assume 25,75 is given as input. The result of the transformation is shown in Figure 3.

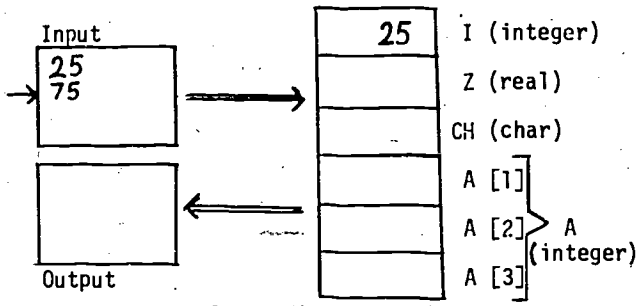


Figure 3. Transformed Memory Model.

This and any other read statement is executed by displaying the transfer of an item from the input box to the proper box of the memory. Note that the item in the input box is not erased, only its copy goes to the memory. At the same time, a small arrow which indicates the beginning of the input queue is moved to the next element. It helps the student to determine what element is to be read next. Transfer of the item to the memory causes the erasing of the previous value. If the type of the item in the input box does not match the type of the memory box, the transformation of the model is aborted and the model returns to its previous state. In such a case, an error message is displayed. The latter rules apply also for the assignment statement which causes some computations and transfers the result to the specified box in the memory. The write statement transfers the proper values from the memory to the output box. Output is treated like a mini CRT screen using a scrolling mode. Every transfer "from" is executed by making a copy and transferring it to the proper box. Various ways of animation of the transfer were considered. Blinking and/or showing characters moving on the screen were chosen as the most explicit approach.

One of the areas where students have many problems and need more assistance and opportunity to practice is in the concept of pointers. Unfortunately in this case, the model of the computer changes during the execution of the program, so we cannot directly use the previous two step method. To make the simulation feasible, we assumed some restrictions for the pointers. The following declaration is assumed:

```
Pointer = ^ Item;
Item    = record
            Key: integer;
            Next: pointer
            end
```

For example, assuming the declarations P,Z: Pointer the following sequence of instructions would result in the picture shown in Figure 4.

```
new (P) ;
P ^.Key: =7;
new (Z) ;
Z ^. Key: = 9;
P ^. Next: =Z;
```

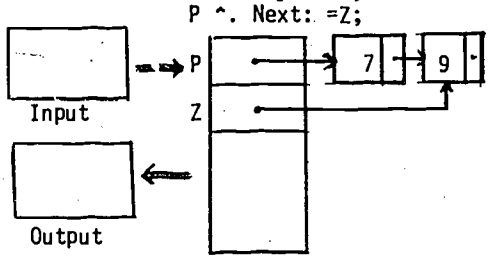


Figure 4. A Memory Model for Pointers.

We are convinced that the above described memory model enables a down-to-earth interpretation of single instructions (as well as their sequence) as a "visible" transfer of some values from some boxes to other boxes and the creation of linked lists. In the model, the number of variables and the size of the arrays and the number of elements in linked lists are restricted because of the size of the screen. To make our model more usable we can relax these restrictions assuming that the screen is the window to display the first elements only.

3. THE PROGRAM GENERATION

Research in automatic programming has achieved some success with experimental systems which produce programs from some specification[5]. Various specification methods have been used[5]:

1. formal
2. by examples
3. natural language

In this paper we describe a simulation model to generate programs found in introductory courses of programming languages e.g.[4-6]. A variety of structures of the programs were analyzed and evaluated for potential generating. To simplify the generating process, we restrict the area to programs with one loop or without any repetition. Exceptions to this rule were programs with additional initialization loop for the array. This restriction is not really very strong because we can almost always break the problem into subtasks in a reasonable way, and use the system to generate every subroutine separately.

Our method of specification is based on choosing options of the given menu. Some restricted formulas are also allowed. External specification corresponds to the internal representation in the form of a network, which is the basis for the pro-

gram generation. The method of generation is based on the observation that most programs can be divided into areas which are independent of each other.

In the first step we need to choose a proper structure for these areas. The following structures are available:

1. No-Loop
2. For-Loop
3. Sentinel_Loop
4. Eof/Foln_Loop
5. Mixed_Cond_Loop
6. Nested_Loop

For example, the For-Loop structure is shown in Figure 5:

```
Program

var

    I:integer;

begin
                                        Initial
                                        Processing
for I:=1     to  100  do

    begin
                                        Read
                                        Area
                                        Processing
                                        Area
    end;
                                        Final
                                        Processing
end.
```

Figure 5. Initial For_Loop Structure

Once the structure is generated (e.g. for_Loop in Fig.5) we decide whether reading inside the loop is necessary or not. If our answer is "yes", we have to list all items to be read together with their type. For example the result of choosing to read one integer variable TRANS is shown in the reading area in Fig. 6.

The third step enables us to specify the processing (including initial processing). We can choose between various standard processes like accumulating the sum, counting, etc. If we decide to have a standard computation such as counting the number of transactions greater than 50, then our final program would look like Figure 6.

```
Program Count (input, output);

var

  I,X,Count:integer;

begin
Count : = 0;                            Initial
                                        Processing

for I = 1 to 100 do

   begin
   read (TRANS);                        Read
                                        Area
   if TRANS >50 then                    Processing
      Count:=Count + 1;                 Area
   end;

write ('Number of items = ', Count);    Final
                                        Processing
end.
```

Figure 6. Generated For_Loop Structure

4. THE FLOWCHART GENERATION

The evolution towards the use of higher level structures has reduced the need for detailed flowcharts. Flowcharting might still be useful, however, for the beginners to visualize control flow. Graphics primitives corresponding to three basic control structures, (sequence, condition, loop) were designed to implement this model. On this basis, a flowchart can be created for the given:

1. program specification
2. program

Process of flowchart creation from the program specification is analogous to the three step program generation. The flowchart generated for the example from the section 3 is shown in figure 7.

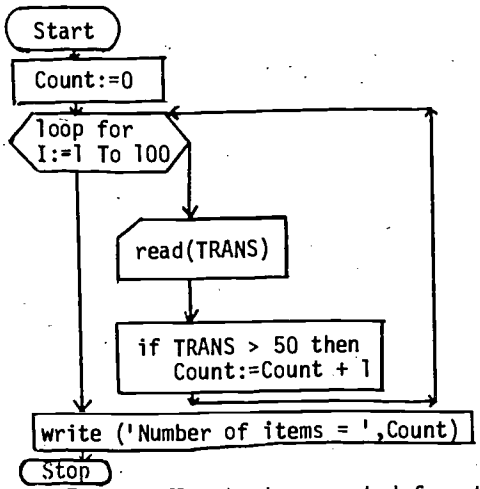


Figure 7. The flowchart generated from the program specification

In the current system any decision structure is included in the processing box to stress the control flow for the loop. The other reason for making this decision is the fact that the size of the screen would impose some restrictions for the number of explicit decision boxes.

The second option is to generate the flowchart directly from the program. In this case, the system displays the flowchart in a more "traditional way" as it is shown in Figure 8. This flowchart is generated directly from the program in Figure 6.

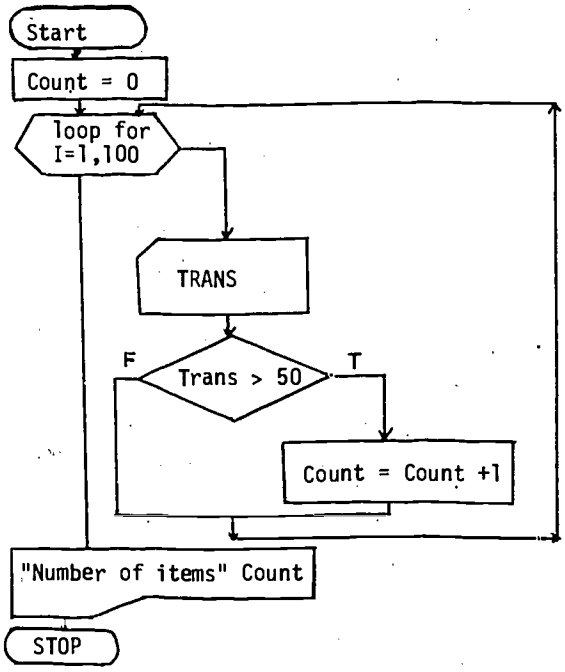


Figure 8. The flowchart generated from the program

5. THE SIMULATION MODELS IN A CAI SYSTEM

The primary purpose of developing the models described above was to use them in a CAI system to teach programming languages. Three modes of interaction with the student for the models were identified:

1. Information
2. Simulation
3. Testing

The Information mode is simply presenting the work of the simulation model. The necessary input is prestored by the instructor. In the case of the memory model, the instructor needs to store the program and the data. For the program generator, the problem description and the program specification should be prestored. For flowcharting, the program should be provided or the problem description and the program specification should be given. The second mode is simple simulation and is similar to the "Ten Finger Exercise"[3]. It allows the students to experiment with the language. One can provide programs or data and watch the changes in the memory and output (the memory model). One can supply specifications of the program mainly by answering the questions and observing the process of generating the program (the program generator). One can also experiment with the flowchart (the flowchart generator). The testing mode is to check whether students can perform actions indicating their educational achievements. They can be asked to predict the changes of the content of the memory for a given program and some data (prestored by the instructor) using the memory model. They can attempt to complete the program or the flowchart for the given problem (provided by the instructor). The system should compare the answers with those generated.

We plan further investigation of the following extensions to the models and their use in CAI:

1. Implement the testing mode
2. Integrate the simulation system so that a sequence of simulation models can be invoked according to figure 1.
3. Give the instructor or the student the possibility to define their own language for a memory model.
4. Build a memory model for the lower level languages.
5. Design educational games (eg adventure games) connected with the programming process.
6. Use or design the interface to extract program specifications from the problem description given in a natural language.
7. Extend the area covered by the program generator.

SUMMARY

In this paper we described the use of simulation models in teaching programming languages. We identified and defined the memory modeling the program generation and the flowchart creation. We showed how to generate and transform pictorial structures using these simulation tools. The application of these models to computer-assisted instruction was shown. Finally, further research in the area was suggested.

REFERENCES

1. Czejdo, B.,"Using Model-Based Instruction to Teach Pascal" Proceedings of NECC/2, Norfolk, Virginia, 1980.
2. Kolkowski, L.,Nauczanie Problemowe w Szkole Zawodowej, Warsaw: WSIP, 1974.
3. Bork, A., Learning with Computers, Bedford, MA: Digital Equipment Corporation, 1981.
4. Richards, J., Pascal. New York: Academic Press, 1982.
5. The Handbook of Artificial Intelligence, Vol.2, Stanford, William Kaufmann, Inc., 1982.
6. Friedman, F., Koffman E., Problem Solving and Structured Programming in FORTRAN. Reading, Massachusetts: Addison-Wesley, 1981.
7. Czejdo, B., "Transformation of Universal Algebraic Expressions in PASCAL" ACM Computer Science Conference, Kansas City, Missouri, 1980.

Raymond E. Bigliani
Lewis Dove
Stephen Bryant
H. Herbert Edwards
Kenneth Keudell
P. James Nielsen
Gerald White
Robert W. Henkens
David Alexander
Richard Cornelius

ABSTRACT: The Use of an Apple/Corvus Networking System in an Elementary Physics Course

Raymond E. Bigliani, Associate Professor of Physics, State University Agricultural and Technical College, Farmingdale, NY 11735

Course Description. A network of eleven Apple microcomputers was used in a one semester elementary physics course (PH095) offered at SUNY Farmingdale in the Spring 1981 semester. This course is taken by students in the Pre-Engineering Technology Curriculum which is designed to help high school graduates satisfy requirements for admission into a two year Engineering Technology program at SUNY Farmingdale. The elementary physics course described below is designed to prepare "pre-tech" students for a one year physics ourse taken in an Engineering Technology curriculum.

I taught two of the four course sections using the microcomputer system described below; the remaining two sections were taught in a traditional, non-computer mode.

Course Objective and Structure. The major objective of the course was to develop an understanding of the basic concepts and to provide drill in the following areas: 1) mathematical skills, 2) graphical data analysis, 3) problem solving technique, and 4) physics theory.

The course was structured as follows: classes met for three one-hour classes per week. Two of the three classes met in the Computer Aided Learning Laboratory (CALL) which consists of eleven Apple microcomputers networked together using a Corvus networking system with a 10 megabyte hard disk drive. The third class met in a regular classroom in a traditional lecture class format.

All homework assignments, called activities, were first disussed in the lecture class; the actual performance of the activity and/or presentation of data occurred in CALL. Some of the activities consisted of drills in the following areas: calculator operation, algebra, graphical analysis, and trigonometry. Seven additional physics content activities were used; these consisted of four to six homework problems taken directly form the text used in the course. Although the wording for each problem was the same for every student, any required numerical values were randomly generated and the corresponding correct answer was internally computed. Students were allowed up to three attempts to correctly answer the question before the correct answer was displayed and the student was then directed to the next problem. If the student answered the problem correctly in three or less tries, he then was congratulated and automatically cycled to the next problem. All questions in the activity had to be answered before a grade could be assigned. In addition, in order to raise their grade, a student could repeat an activity as many times as desired. However, only the last grade on an activity was retained.

Course Evaluation. The course was evaluated using the results of a student qeustionnaire and standardized post-tests administered to all students enrolled in the course. Student response to the microcomputer as a testing tool was overwhelmingly positive, mainly due to the ability to answer an individual question up to three times and the ability to repeat any activity any number of times.

At the end of the semester, all students took College Board Entrance Examination tests in Computation and Elementary Algebra. The two "computer" classes averaged slightly higher on both tests than did the "non-computer" classes. A more meaningful measure of the efficacy of the technique described above will be determined from a future correlation study between the "computer" and "non-computer" groups in PH095 and their grades in their subsequent physics courses.

ABSTRACT: Program Development by a Biology User's Group for Microcomputer Assisted Instruction

Lewis Dove, Stephen Bryant, H. Herbert Edwards, Kenneth Keudell, P. James Nielsen, Gerald White, Western Illinois University, Macomb, IL 61455

A biology user's group has been operating at Western Illinois University for two years. It is composed of six faculty, five from the Department of Biological Sciences and one from the Mathematics Department. The Project Director, a cell biologist, spent two years promoting CAI and computer literacy as an associate of the faculty development office. Other members of the group include a plant pathologist, a microbiologist, a human physiologist, a population biologist, and a

mathematician who teaches computer programming. A grant from the Apple Foundation provided two microcomputer systems, a faculty development grant provided funds for the development of an immunology program, and the remaining rpograms were developed under a grant from the National Science Foundation Local Course Improvement (LOCI) program. The group promotes computer literacy and microcomputer usage among biology students and faculty by reviewing microcomputer software in biology, by presenting a course on microcomputers in biology, and by writing software. Five of these programs are being evaluated by students and others are being written. These and other biology programs acquired from external sources are being used by students to supplement their laboratory work in first-year biology courses, general microbiology, ecology, genetics, and cell biology. These include a program which assists sewage treatment personnel in evaluating sewage effluent by providing a key to indicator protozoa and a description of their taxonomy. Another program dramatizes electron flow in light reactions of photosysnthesis and also includes the dark reactions. The immunology tutorial program describes and illustrates the immunoglobulins. A tutorial microorganism identification program characterizes the bacteria, fungi, algae, and protozoa. A computer-managed program written in Pascal incorporates three lessons in cell biology and a student file system. Projects under developmnent include energy metabolism of mitochondria, adaptations of existing software in chemistry to courses in biology, and tutorial/simulations of natural selection and migration.

ABSTRACT: Scientific Instrument Trainer

Robert W. Henkens, P. M. Gross Laboratory, Duke University, Durham, NC 27706

Rationale. Access to necessary instruments for work at the frontiers of knowledge is obviously important to the nation's future research productivity and for the training of future generations of scientists and engineers.

Most science educators agree that laboratory training is necessary for science students. Unfortunately, high costs severly limit this kind of training in many areas of instrumental analysis, including modern spectroscopic analysis, This is in strong contrast to a decade ago when most universities provided their students with hands-on experience with state-of-the-art instrumentation.

The Scientific Instrument Trainer System is designed to help with this problem by providing interactive training for chemistry graduate students in modern FT-nuclear magnetic resonance (NMR) instrumental analysis. The system could also be used in undergraduate upper division chemistry laboratories. The basic system architecture is general and might be adapted to other scientific instrumentation.

Expected Product. The main components are an instrument simulator for the student to use, and an instrument communications package for the instructor. The system will provide an interactive FT-NMR simulation with extensive CRT graphics for the Apple II computer. The Trainer is meant for individual use in a course or training program. A unique aspect of the product is the provision for data communications with the instrument itself, allowing the instructor to download a number of real data sets of 4-8k each. The concept should be adaptable to a wide range of additional scientific instruments.

The input and output characteristics of Scientific Instrument Trainer will be like the instrument, using keyboard, display screen, printer, and plotter. The students can set parameters, such as the accumulation time, pulse angle, and decoupler power and frequency, and give commands through the keyboard, obtain parameter listings, and observe instrument graphic displays. At the end of the training session, the student can go over printed parameters and plotted data with his instructor and discuss possible ways for improvement and return to the trainer to gain experience and confidence to handle a variety of experimental problems.

ABSTRACT: Concentrated Physics Concepts: A Comprehensive Package of Tutorial Problem Solving

David Alexander, Physics Department, Richard Cornelius, Chemistry Department, Wichita State University, Wichita, KS 67208

Undergraduate students enrolled in an introductory physics course often experience difficulty relating abstract physical principles in the concrete problem-solving arena. In many cases, this difficulty arises from a deficiency in general problem-solving skills. Unfortunately, undergraduate physics classes are often too large for the instructor to give students the individual assistance needed to develop the logical thought processes required for successful problem solving. Although laboratory experiments are an important element in learning physics, they address a different set of goals.

Concentrated Physics Concepts is a package of Apple II+ computer programs designed to foster the development of problem-solving skills for introductory physics students. The programs are contained on about eight diskettes and cover the material normally contained in a two semester introductoery physics course. Two types of programs are included: Conceptual programs review the concepts and terminology of a unit to assure that the student understands the material before attempting the problems associated with the unit. Each problem-solving program presents a graded sequence of problems to illustrate a single physical concept. In many cases, the solution of the problem is approached through several preliminary questions. When requested, the programs provide assistance in working out an appropriate strategy for solving the problem.

These programs incorporate both user friendliness (no prior computer experience is required to operate any part of the package) and user power (the student controls the pace and subject matter at all times). Whenever possible, student interaction with the screen (e.g., properly placing vectors on a diagram) is incorporated into the problem solution. Problem solving techniques are emphasized by asking appropriate preliminary questions and by providing assistance in response to incorrect answers.

COURSEWARE DEVELOPMENT FROM THE PUBLISHER'S PERSPECTIVE
A PANEL DISCUSSION

M. D. Roblyer, Moderator
ICON Enterprises

Jack Chapel
SRA, Inc.

Harvey Guion
Random House Inc.

Jane Isay
Harper and Row

Dale LaFrenz
Scott, Foresman Co.

Christine Johnston
Milliken Publishing Co.

Kenzi Sugihara
Harcourt Brace Jovanovich, Inc.

As publishers of educational materials continue to expand their development of computer-based instructional products, the role of these organizations in shaping the instructional computing field is becoming increasingly apparent. In this panel presentation, high-level representatives from several major publishing houses will discuss and answer questions on some of the issues surrounding their involvement in educational technology. A specific focus will be on the methods publishers use to assure that materials they publish are: (1) responsive to the needs of educators, (2) helpful to the continued evolution of the field, and (3) of high instructional quality.

Trends in Interactive Data Analysis In the Classroom

Jon A Christopherson, Chair
U.S. Coast Guard Academy
New London, CT. 06320

SPONSOR: SIGCUE

ABSTRACT

The course Social Science Methodology at the U.S. Coast Guard Academy tries to sensitize the cadets to the nature of applied statistical analysis. The tedium of computing one Pearson's r correlation coefficient over 40 cases is usually enough to convince the cadets that they should learn to use a computer system. While not course materials for classroom use are designed for batch processing, batch processing is pedagogically an undesirable teaching tool. The central problem lies in the loss of continuity in the teaching of a concept. The best approach is to use an interactive data analysis system (IDA). The desirable characteristics of a good IDA system include the ability to:

1) create data files either by keying in the data or by using a prepared data file,

2) randomly access any of the variables in the data base in any order,

3) dress up the output with labels for variables and values,

4) edit online any of the variables in the data base after the variable has been created,

5) automatically handle missing value codes specified for any variable,

6) create data (or conditionally create) and perform general mathematical manipulations of variables and numbers, (only one topic in the Methods course) is taught in an interactive mode. We find that cadets more easily learn the basic underlying logic of the method and the practical results of the violation of regression's assumptions. An interactive data analysis system allows the instructor to follow the logic of class discussion rather than having it dictated by fixed printed material.

Michael Smith will discuss an elective course in the use of computers in social welfare research which has been developed for students in the doctoral program at Hunter College School of Social Work. The purpose of the course is to expose doctoral students who are not research majors to basic data analysis procedures in social research, to give them an understanding of statistical and analytic procedures and to help them learn a set of skills which can be used directly in their doctoral projects. An experimental approach to data analysis contains learning that can never be realized in a survey course with a lecture approach. This paper describes three basic issues in the course: 1) the initial attitude of students; 2) the stress given to the role of data analysis in the total research process; and 3) the choice of a packaged program.

The third presentation will discuss micro versus mainframe conversational computing. With the advent of microcomputers, students approach the computer with less trepidation. As micros become more powerful, colleges must make a choice between offering students computing in the friendly micro environment or encouraging conversational mainframe computing.

The IDA Interactive Data Analysis and Forecasting System, a teaching and research tool which runs on both mainframes and micros is used as a case study.

PARTICIPANTS:

Joan Fee
SPSS, Inc.
Chicago, IL 60611

Michael Smith
Hunter College
New York, NY 10021

Loren Bullock
IBM Corporation
Bethesda, MD 20817

William Gattis
The Tandy Company
Fort Worth, TX 76102

Science Education and the Growth of the U.S. Computer Industry: Is This a National Policy Concern?

Dorothy Derringer, Chair
National Science Foundation

ABSTRACT

The U.S. computer industry shows a strong positive balance of payments and is one of the few growth areas in the economy. Many feel that a scientifically and technologically literate workforce is key to increasing the growth of this industry. Should the government act to create a climate which will encourage this industry to thrive? If so, how should this be done? Panelists will discuss answers to these questions within the context of the economy, the role of education in fostering the industry, and government state actions. Differing views will be presented on proposed legislation and possible actions.

PANELISTS

Vico E. Henriques
CBEMA

Fred Weingarten
Office of Technology Assessment

N. John Castellan Jr.
Indiana University

William Aldrich
National Science Teachers Association

SPONSORS

SIGCUE
ICCE

Computing Curricula Prepared by Professional Societies

Joyce Currie Little, Moderator
Towson State University

SPONSOR: SIGCSE

ABSTRACT

Curriculum recommendations for the education of computer professionals have been developed and published by several United States computer associations, including the Association of Computing Machinery (ACM), the Data Processing Management Association (DPMA), and the Computer Society of the Institute of Electrical and Electronic Engineers (IEEE-CS). Among international groups, the International Federation for Information Processing (IFIP) has produced one report, now being revised. The American Federation of Information Processing Societies (AFIPS) serves as liaison to IFIP on behalf of its United States member associations.

This session will provide an overview of these curriculum reports. A bibliography, with other information, will be available. Included are works for secondary schools, vocational technical institutes, community colleges, baccalaureate degree programs, and graduate programs.

Representatives from each association will discuss the level and type of institution for which their works are intended, the subject matter content recommended for the program, and the type of qualifications for jobs or further study expected after completion of the program.

Opportunity will be given to each association to acquaint the audience with their current and ongoing curriculum development work.

PARTICIPANTS:

ACM: Richard Austing, University of Maryland
Gerald L. Engel, Christopher Newport College

DPMA: David Adams, Northern Kentucky University

IEEE-CS: J. T. Cain, University of Pittsburgh
Murah Vuranasi, University of South Florida

IFIP: William F. Atchison, University of Maryland

Augmenting Self-Study Materials With Microcomputer-Based Lessons: A Case Study

Ernest Giangrande Jr.
Department of Computer Science
North Adams State College
North Adams, MA 01247

William S. Bregar
Computer and Information Sciences
University of Delaware
Newark, DE 19711

Abstract

The central question addressed in this paper is the effectiveness of computer-based instruction in a self-study, self-paced learning environment designed to aid students learning the syntax and semantics of FORTRAN input and output statements and FORMAT statements. Our results indicate that CAI lessons can have a noticeably positive effect on learning, and there is evidence that this effect can be attributed to CAI, rather than to either a novelty effect or the fact that the materials augmented basic coursework.

Introduction

The effectiveness of CAI studies are generally open to the charges that 1) there is a novelty effect associated with the use of the computer as the instructional medium, and 2) the differences attributed to CAI may, in fact, be due to individualized self-study. Our study attempts to neutralize these factors by comparing CAI lessons to self-study materials for a course in FORTRAN developed for the Department of Computer Science at Oregon State University. Students in a Computer Science course should be less affected by the novelty of using a computer, and the self-study aspect of the particular course compares with the individualized nature of CAI. A welcome side effect of this study was the development of a usable production level CAI program for teaching the difficult topic matter involved in the FORTRAN FORMAT statement. The materials were designed to be supplemental in nature, rather than as a stand-alone replacement for the course materials. Our experimental procedure also takes this into account.

The limited number of publications comparing CAI to self-study materials have generally reported contradictory results (see Bailey and Klassin, 1979; Splittgerber, 1979; Edwards 1978; Hazen et. al., 1979; and Chambers and Sprecher, 1980).

Background

As mentioned, the basis for this study was the self-study course (CS 199, Self-Study Introduction to FORTRAN Programming) at Oregon State University (O.S.U.). To successfully complete this course students must pass (70%) nine quizzes and satisfactorily complete (60%) six programming assignments. Students are expected to demonstrate competency in all areas covered in the course, therefore, they are allowed to retake quizzes and resubmit program assignments until the course criteria are met. The materials used by the students include a set of course notes developed for this course and a textbook (Krutzberg and Schneiderman, 1975). There are nine lessons outlined in the course notes. Quizzes are administrated by the Math Science Learning Center (MSLC) at O.S.U.

Grade records in this course indicate that the percentage of students successfully completing this course has ranged between 50 and 60 percent Furthermore there are a few key areas where students have difficulty, as determined by the number of retakes for quizzes in these areas. Clearly, lessons with a high number of retakes contained material that was difficult for the students to master. An analysis of quiz results showed that two primary areas of difficulty were Input/Output statements and looping. Lesson 4, which introduces formatted input/output (the focus is on the I, F, X, and string specifications) was selected for our experimental target. CAI lessons were prepared to supplement the existing materials. They were written in Pascal for the Apple II.

Lessons for FORTRAN I/O:

CAI Lessons

Two supplemental lessons were developed for the study. One was the CAI lesson, the other was a printed lesson essentially duplicating the CAI material. This was to ensure that the effect observed from the use of the CAI lesson would not be attributed to its use as a supplement to the original course material.

The primary focus of the lesson is the construction of READ, PRINT, and FORMAT statements in FORTRAN. The following concepts must be clear to a student attempting to understand FORTRAN I/O:

a) the relationship between a variable name and a location in a computer's memory.

b) the relationship between the name chosen for a variable and the type of data that is stored in its location;

c) the difference between the data types that can be processed using a FORTRAN program.

The CAI lesson has five parts, each containing textual material with examples or problems for practicing the techniques covered. Following is a list of these five parts and a brief outline of their content:

1) READ/PRINT-CONSTRUCTION
   - basic concepts related to input and output
   - rules for constructing READ and PRINT statements
   - examples of READ and PRINT statements
2) READ/PRINT-PRACTICE
   - student constructs READ and PRINT statements from specifications
   - system evaluates student's response and gives feedback
3) FORMAT-CONSTRUCTION
   - rules for constructing FORMAT statements used for input
   - rules for constructing FORMAT statements used for output
   - examples of FORMAT statements used for output
4) READ/FORMAT-PRACTICE
   - student constructs FORMAT statements used for input from specifications
   - system evaluates student's response and gives feedback
   - system simulates execution of input statement using student's FORMAT
5) PRINT/FORMAT-PRACTICE
   - student constructs FORMAT statements used for output from specifications
   - system evaluates student's response and gives feedback
   - system simulates execution of output statement using student's FORMAT.

The READ/PRINT-CONSTRUCTION part and the FORMAT-CONSTRUCTION part present the basic concepts for constructing the READ, PRINT, and FORMAT statements. Examples are presented along with a simulated execution of each.

The practice parts are the primary focus of this lesson. The READ/PRINT-PRACTICE part provides an opportunity to construct READ and PRINT statements from specifications and to have the system evaluate the response. The student's task is to enter the appropriate I/O statement. The system evaluates the response in terms of spacing, presence of required keywords and FORMAT statement number, and correct choice of variable names. Errors are described to the student. If the response is correct then appropriate specifications are presented in the FORMAT statement and either a data card or values stored in memory are supplied. The system then simulates the execution of the statement, processing one specification at a time.

It should be noted that the system randomly produces all specifications and data values used in these problems and those discussed below. The system generates all statements using the syntax rules for their construction. The type and number of specifications are determined randomly but there is no template for the statement that is filled in by the system. This approach differs from generative techniques (Collins and Duff, 1979; Garcia and Rude, 1979), and is similar to that of Koffman (1972).

The READ/FORMAT-PRACTICE part and the PRINT/FORMAT-PRACTICE part give the student an opportunity to construct FORMAT statements for READ and PRINT statements respectively. The specifications needed to construct the FORMAT statements are presented to the student who then enters the FORMAT statement at the keyboard. The student's response is evaluated in two steps. First, the syntax of the response is evaluated. Any errors detected at this stage are presented to the student along with a correct FORMAT statement. The student is then presented another problem.

If the response contains no syntactic errors it is semantically evaluated. This evaluation can result in one of two outcomes: (1) the response is semantically correct and will produce the specified result, or (2) it is semantically incorrect and will produce some undesired result (in the case of input incorrect data values will be read from the card, and for output an incorrectly structured output line will be produced). The system presents the results of this evaluation to the student.

Regardless of the result of this evaluation, the system will simulate the execution of the student's response similar to the way described for the READ/PRINT-PRACTICE part. The simulated execution of the student's semantically incorrect FORMAT statement is carried out to demonstrate what an erroneous FORMAT statement would actually produce. The student is then shown a correct FORMAT statement and its results. Thus, the student is shown what an incorrect FORMAT specification would produce and its effect on the rest of the items being read or written.

The Printed Version of the CAI Lesson

To insure that any observed effectiveness of the CAI based materials was, in fact, more likely a function of the medium and not the content of the lesson, a comparison in the form of a printed lesson was also developed. This printed lesson contains of all the textual material presented in the CAI version including all of the examples from the CAI version. Students using this version would not be presented problems to solve and have no opportunity to practice the skills presented in the lesson.

Methodology

The design consisted of three groups: Group 1 (CAI-EXP), an experimental group that used the CAI lesson, Group 2 (PRT-EXP), an experimental group that used the printed lesson, and Group 3 (CONTROL), a control group that used no supplemental materials.

Subjects

Subjects were drawn from students enrolled in CS 190, Self Study Introduction to FORTRAN Programming, during the winter term of 1981 at Oregon State University. During the orientation meeting the students were asked to participate in a project that they were told was designed to evaluate their impressions of new materials designed for this course. Each student decided whether he would volunteer. Twenty of the volunteers were randomly selected and assigned to each of the experimental groups (CAI-EXP and PRT-EXP). The remaining volunteers were assigned to the CONTROL group. Since twenty students did not remain for this group it was supplemented by randomly choosing students enrolled in the course who had not volunteered for the project.

The students were told to contact the instructor after completing all of the standard materials for this lesson but prior to taking quiz 4. At that time each student was informed of the group he was in and how to access the appropriate materials. He was also told that he would be given a questionnaire to complete regarding the lesson he viewed. Volunteers in the CONTROL group did not use any supplemental materials and did not complete a questionnaire. They were told that their assistance was not needed because there were too many volunteers and to continue with the course as usual.

Procedure

Upon contacting the instructor each student was informed of the type of material he would be viewing and told how he could access it. If he was assigned to the CAI-EXP group, arrangements were made for him to meet with an assistant who directed him to the microcomputer lab and stayed with him until the lesson was completed. The assistant provided the lesson disk and printed instructions. He then went about his own work but was available if the student had any questions. Upon finishing the lesson a questionnaire was completed.

Students assigned to the PRT-EXP group, were directed to the resource desk of the Math Science Learning Center at O.S.U. where they received the printed lesson and a questionnaire. Both were returned when completed. The volunteers assigned to the CONTROL group were told they were not required to participate in the study and those who had not volunteered never contacted the instructor.

The number of students dropped from the original twenty per group. This was caused by two factors: (a) only those students who remained active in the course through lesson 4 and took the quiz at least once, were considered, and (b) some students in the CAI-EXP and PRT-EXP groups asked to be excused from participating in the study. These students were dropped from the study since they had not been exposed to the treatment before they took the quiz for this lesson. The groups eventually ended up with 14 students in the CAI-EXP group, 16 in the PRT-EXP group, and 17 in the CONTROL group.

In an attempt to demonstrate that the groups were homogeneous two items from the questionnaire were examined - student G.P.A. and prior experience with computers. These tests could only be applied to the CAI-EXP and PRT-EXP groups as they were the only ones who filled out the questionnaire. An analysis of variance applied to the G.P.A.'s for the members of these groups (see Table 1) shows no significant difference between them on this measure ($F = 0.222$, $p > .05$). A Chi square test showed no difference between these groups in terms of prior computer experience.

The only measure that could be used to confirm the homogeneity of all three groups was student classification (i.e. freshman, sophomore, etc.). A Chi square of 10.59305 ($p > .05$) suggests that there is no difference between groups on this measure.

Results

Data pertaining to quiz 4 were analyzed with respect to the number of retakes, the total scores, and the number of correct answers on the 15 I/O related questions. Quiz 8, which also tested I/O concepts was correlated with quiz 4.

The measures used were analysis of variance (ANOVA), Chi square, and Pearson's correlation. The analysis of variance was also used as an appropriate follow up test to investigate any variance found between groups (Woca, 1977). All of the analyses were done using SPSS - A Statistical Package for the Social Sciences. A $p < .05$ was required before the null hypothesis was rejected.

A Chi square of the number of tries at quiz 4 by group showed that there was a significant difference ($p < .05$) in the number of tries between the groups. No member of the CAI-EXP group took a retake on this quiz while 50% of the PRT-EXP group and 76.5% of the CONTROL group took one or more retakes.

An analysis of variance was conducted on the number of correct responses to the I/O questions on the first attempt at quiz 4 by the members of the three groups (see Table 2). A significant F ratio ($F = 10.66$, $p < .05$) suggests that there was some variance between the groups on this measure. The follow up tests used to isolate this variance (Table 3) shows a significant difference ($F = 12.401$, $p < .05$) between the CAI-EXP group and the PRT-EXP group on this measure. It also shows a significant difference ($F = 20.829$, $p < .05$) on the measure between the CAI-EXP group and the CONTROL group. Since there was no difference ($F = 1.538$, $p > .05$) between the PRT-EXP group and the CONTROL group on this measure, the observed variance can be attributed to the effect from the CAI-EXP group's performance.

A similar analysis was applied to the total scores for the first attempts at quiz 4 by the students in the three groups. Table 4 suggests that some variance exists between the groups on this measure ($F = 9.60$, $p < .05$). The results of the follow up tests used to isolate this variance (see Table 5) shows a significant difference ($F = 14.352$, $p < .05$) between the CAI-EXP group and the PRT-EXP group. There is also a significant difference ($F = 17.475$, $p < .05$) between the CAI-EXP group and the CONTROL group. An F ratio of 0.204 ($p > .05$) between the PRT-EXP group and the CONTROL group was not significant. These analyses suggest that the variance can again be attributed to the CAI-EXP group's performance.

Since quiz 8 also deals with Input/Output a Pearson's correlation was applied to the I/O scores and total scores for that quiz and quiz 4. The results, ($r = .3852$, $p < .05$) and ($r = .3852$, $p < .05$), respectively indicate that these factors correlate with their counterparts on quiz 4.

Discussion

The performance of the PRT-EXP group was not found to be significantly different from that of the CONTROL group. The CAI based materials had an impact on the the performance of the CAI-EXP group. Both groups were exposed to the same textual material, yet no student in the CAI-EXP group had to retake the quiz for lesson 4, while 50% of the PRT-EXP group took at least one retake. Significant differences in performance on the related questions and total quiz scores support this conclusion. It cannot be concluded that the content of the presentation is responsible for the observed effect. One can only conclude that the CAI-EXP group's performance was influenced by the opportunity to do practice problems in an enriched environment, that is, in an environment where the system analyzed responses and provided immediate feedback.

Splittgerber (1979) reported that the usefulness of simulation and problem solving is questionable in CAI lessons. The findings reported here are contrary to his. The only significant difference between the textual materials and the CAI lesson developed for this study was the inclusion of problem solving and dynamic simulation in the CAI lesson. It is likely that the reported differences in the effectiveness of these techniques is due to the environments in which the lessons were used.

It might be argued that programmed instruction provides a similar environment to that provided in a CAI environment. This is not the case when the CAI environment provides simulation. The lesson used in this study not only provided immediate feedback tailored to the student's response but also provided a simulated execution of input and output statements. Simulations might be provided in a programmed text but the examples must be predetermined by the author. The ability to provide simulated execution of a student's response can only be provided in a one-to-one teacher student situation or in a CAI lesson like the one developed here.

The attitude differences reported by Crawford (1970) and Edwards (1978) for students using CAI based materials can be discounted in this study since all students involved were exposed to computers as part of their course requirements. Hazen (1979) reported no attitude differences in a study of a FORTRAN programming course for business students, supporting the position that the observed effect was due to the aspects of the lesson and not to some effect resulting from exposure to the computer itself.

Our findings suggest that the use of CAI as a supplement to existing self-study materials warrants further study. Similar lessons for other problem areas in FORTRAN programming should be developed and the effect of their use studied. CAI lessons should also be developed to supplement self-study courses in other disciplines to determine if the effect reported here would also be found in non-computer programming environments.

Bibliography

Bailey, D.E. Ingredients for Excellence in Computer-Based Education Systems. In National Educational Computing Conference, 1979, University of Iowa, pp 2-6.

Chambers, J.A. and Sprecher, J.W. Computer Assisted Instruction: Current Trends and Critical Issues. Communications of the ACM, 1980, 23,6, pp 332-342.

Collins, R.W. and Duff, S.J. Computer-Assisted Test Construction via Automatic Program Generation: Using PROBGEN II to Create Individualized Exams and Problem Sets. In National Educational Computing Conference, 1979, University of Iowa.

Crawford, A.N. A Pilot Study of Computer-Assisted Drill and Practice in Seventh-Grade Remedial Mathematics. California Journal of Educational Research, 1970, 21 pp 170-181.

Edwards, L. The Effects of CAI on Achievement and Attitude in the Freshman Survey Mathematics Curriculum. In Ninth Conference on Computers in the Undergraduate Curricula, 1978, University of Denver, pp 16-23.

Garcia, A and Rude, S.A. Design of a CAI Tool to Generatively Teach Ecology. In National Educational Computing Conference, 1979, University of Iowa.

Hazen, M., Daly, C., Embley, D., Nagy, G., and Prange, W. Initial Evaluation Results for an Introductory Programming Course Without Lectures. In National Educational Computing Conference, 1979, pp 150-160.

Koffman, E. B. A Generative CAI TUtor for Computer Science Concepts. Proceedings of the 1972 Spring Joint Computer Conference, 1972, pp 379-389.

Splittgerber, F.L. Computer-Based Instruction: A Revolution in the Making? Educational Technology, 1979, 19,1, pp 20-25.

Wood, G Fundamentals of Psychological Research. Boston: Little Brown and Company, 1977.

Analysis of Variance

	D.F.	Sum of Squares	Mean Squares	F Ratio	F Prob.
Between Groups	1	.0385	.0385	.222	.6428
Within Groups	20	3.4575	.1729		
Total	21	3.4958			

Table 1.
Anova for G.P.A. for CAI-EXP and PRT-EXP

Analysis of Variance

	D.F.	Sum of Squares	Mean Squares	F Ratio	F Prob.
Between Groups	2	111.4027	55.7013	10.660	.0002
Within Groups	44	229.9165	5.2254		
Total	46	341.3191			

Table 2.
Anova for I/O related questions on quiz 4 for all groups.

Analysis of Variance

	D.F.	Sum of Squares	Mean Squares	F Ratio	F Prob.
Between Groups	1	53.2149	53.2149	12.401	.0015
Within Groups	28	120.1518	4.2911		
Total	29	173.3667			

(a)

Analysis of Variance

	D.F.	Sum of Squares	Mean Squares	F Ratio	F Prob.
Between Groups	1	107.3597	107.3597	20.829	.0001
Within Groups	29	149.4790	5.1544		
Total	30	256.8387			

(b)

Analysis of Variance

	D.F.	Sum of Squares	Mean Squares	F Ratio	F Prob.
Between Groups	1	9.4342	9.4342	1.538	.2243
Within Groups	31	190.2022	6.1356		
Total	32	199.6364			

(c)

Table 3. Follow up tests for I/O related questions on quiz 4: (a) Anova for CAI-EXP and PRT-EXP, (b) Anova for CAI-EXP and CONTROL, and (c) PRT-EXP and CONTROL.

Analysis of Variance

	D.F.	Sum of Squares	Mean Squares	F Ratio	F Prob.
Between Groups	2	2445.8946	1222.9473	9.600	.0003
Within Groups	44	5604.9139	127.3844		
Total	46	8050.8085			

Table 4. Anova for total scores on quiz 4 for all grou

Analysis of Variance

	D.F.	Sum of Squares	Mean Squares	F Ratio	F Prob.
Between Groups	1	1612.6881	1612.6881	14.332	.0007
Within Groups	28	3150.6786	112.5242		
Total	29	4763.3667			

(a)

Analysis of Variance

	D.F.	Sum of Squares	Mean Squares	F Ratio	F Prob.
Between Groups	1	2115.8039	2115.8039	17.475	.0002
Within Groups	29	3511.1639	121.0746		
Total	30	5626.9677			

(b)

Analysis of Variance

	D.F.	Sum of Squares	Mean Squares	F Ratio	F Prob.
Between Groups	1	29.8935	29.8935	.204	.6548
Within Groups	31	4547.9853	146.7092		
Total	32	4577.8788			

(c)

Table 5. Follow-up tests for total scores on quiz 4: (a) CAI-EXP and PRT-EXP, (b) CAI-EXP and CONTROL, and (c) PRT-EXP

The Bridge From Non-Programmer to Programmer

Jeffrey Bonar* and Elliot Soloway**

*Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

**Department of Computer Science
Yale University
New Haven, Connecticut 06520

Abstract[1]

Non-programmers bring to the learning of programming strategies that they have developed to solve day-to-day problems. Interestingly, programming language constructs often require strategies that conflict with these non-programming strategies. One can predict quite confidently that in these situations, novice programmers will have difficulty --- and bugs in their programs will result. In this paper, we present evidence for the existence of *natural language specification strategies* that novices bring to programming, in the form of abstracts from verbal protocols taken from novice programmers as they are trying to program. These transcripts highlight the types the bugs and misconceptions that result when there is a mismatch between the strategies required by programming language constructs and the strategies that non-programmers bring to programming.

1. Introduction

Any interesting computerized task soon involves programming. Experience with statistics packages, word processing, and even microwave ovens shows that we always want our systems to be able to follow a step-by-step specification involving decisions and repeated actions. Even with a very intelligent computerized assistant, we would like to give it detailed instructions at an appropriate level of abstraction.

This ubiquity of programming presents a problem, however. It is widely known that programming, even at a simple level, is a difficult activity to learn. The seventies saw a revolution in the way that programming was practiced and taught. The phrase "structured programming" summarizes a whole new level of attention to the design, implementation, and testing of computer programs; attention that changed much of the thinking about how programming should be taught. We are now much clearer about how to teach powerful and effective programming, but do we know how to make programming maximally available? Do we really know how to make *programmers* ubiquitous? Apparently not:

> *Based on exam grades and on our studies (e.g., Soloway et al. 1982), we estimate that more than 40% of the conscientious students never really understand the rudiments of programming.*

What's missing --- what is the way to build a bridge between non-programmer and programmer?

We begin by noting that programming is a cognitive skill, much like understanding math [Rissland, 1978] or solving physics problems [DiSessa, 1982]. Drawing from recent work in Cognitive Science, we are using a new methodology for looking at the acquisition of cognitive skills. There are two key parts to our methodology. First, we look *in great detail* at the errors of novice programmers. As experienced programmers, our tendency is to look at errorful novice programs only seeking to eliminate bugs as soon as possible. In our work, we have tried to see what was the specific (mis)information used by the novice to produce the bug. This is quite a powerful view: novice programmers have deep and interesting misunderstandings, as you will see below.

Second, in understanding the novice misunderstandings we try to view the situation in terms of specific bundles of knowledge possessed by the expert but not by the novice. What we find is that there is often a level of *tacit* knowledge [Collins, 1978] that is not explicitly taught, and often not even explicitly acknowledged.

In this report we present evidence for one major source of difficulty: *current programming languages often do not accurately reflect the problem solving strategies that non-programmers bring to programming.* That is, non-programmers have developed *natural language strategies* in order to cope with day to day problems. While experienced programmers have learned to modify or replace these strategies with ones more appropriate to computer programming, novices are often confused at this very basic level. Step-by-step natural language specification provides powerful intuitions for novice programmers using a programming language. We hypothesize that these intuitions take the form of bundles of knowledge we call *plans* - regular but flexible techniques for specifying how to accomplish a task. Programming knowledge also involves plans [Soloway et al, 1982] [Waters, 1979]. While an individual programming language plan may have many lexical and syntactic similarities to a corresponding natural language plan, the two plans often have incompatible semantics and pragmatics. Many novice programmer's misconceptions derive directly from these incompatibilities.

[1]This work was supported by the National Science Foundation under NSF Grant SED-81-12403. Any opinions, findings, conclusions, or recommendations expressed in this report are those of the author, and do not necessarily reflect the views of the U.S. Government.

In this brief report we will show examples of natural language plans and programming language plans. We will then analyze transcripts of thinking aloud protocols taken with novice programmers who use natural language plans while attempting to write a computer program. We conclude with a brief discussion of the implications of this work for teaching programming.

Before proceeding, a methodological point is worth making. While the theoretical framework in this paper -- and its conclusions -- are the same as those in other papers we have published (e.g., Soloway et al. 1982, Soloway et al. 1981a), there is a key methodological difference between this paper and the others: previously we used statistical arguments based on written tests with large numbers of programmers as evidence for/against our hypotheses; however, in this paper we use anecdotes from thinking aloud protocols taken from individual programmers as evidence for our hypotheses. That is, in previous papers we have made claims about what we think our subjects were thinking. Statistical evidence is an indirect test of these sorts of claims. Verbal reports of subjects as they are programming provide a more direct window into the thought processes of our subjects. Thus, this paper provides the needed closure for our hypotheses: we have converging evidence from statistical-type group studies and from verbal reports with individual programmers that support our theory of the role of programming plans and the role of natural language plans (i.e., pre-programming plans) in programming.

2. Natural Language Plans and Programs

Consider the following problem:

> *Problem 1*: Please write a set of explicit instructions to help a junior clerk collect payroll information for a factory. At the end of the next payday, the clerk will be sitting in front of the factory doors and has permission to look at employee pay checks. The clerk is to produce the average salary for the workers who come out of the door. This average should include only those workers who come out before the first supervisor comes out, and should not include the supervisor's salary.

The following natural language specification for this problem, written by one of our subjects, is typical:

1. Identify worker, check name on list, check wages
2. Write it down
3. Wait for next worker, identify next, check name, and so on
4. When super comes out, stop
5. Add number of workers you've written down
6. Add all the wages
7. Divide the wages by the number of workers

There are several natural language specification plans used here. Note how steps 1 through 4 specify a loop: steps 1 to 3 describe the first iteration of the loop, indicating repetition with the phrase "and so on". Step 4 adds a stopping condition, assuming that this condition will act as a *demon*, always watching the action of the loop for the exit condition to become true. The specification also assumes *canned procedures* for counting inputs, step 5, and for summing a series of numbers, step 6. Note however, that these two procedures are both denoted with the word "add".

Now focus on the two actions performed in steps 1 and 2. The plan to describe these actions is *get a value (step 1), and process that value (step 2)*. This plan is nearly universal in this sort of description. Unfortunately, many programming languages support a far less natural plan: *process the last value, get the next value*. Below, we discuss this problem in detail.

3. Examples of Novice Programming Difficulties

To show how the conflict in strategies effects novice programmers, consider a problem analogous to problem 1, but simpler and explicitly of a programming nature:

> *Problem 2*: Write a program which repeatedly reads in integers until it reads they integer 99999. After seeing 99999, it should print out the *correct* average. That is, it should not count the final 99999.

In Pascal, a popular novice programming language, the preferred correct solution to Problem 2 is:

```
PROGRAM Problem_2 Expert;
   VAR Count, Total, New : INTEGER;
   BEGIN
   Count := 0; Total := 0;
   Read (New);
   WHILE New <> 99999
     DO BEGIN
        Count := Count + 1;
        Total := Total + New;
        Read (New)
        END;
   IF Count > 0
      THEN writeln ('Average =',Total/Count)
      ELSE writeln ('No data.')
   END.
```

Notice the peculiar WHILE loop construction. Because a WHILE loop tests only at the top of the loop, it is necessary to have a Read both above the loop and at the bottom of the loop. Within the loop we see the plan *process the last value, read the next value*. This plan is part of the knowledge used by experienced Pascal programmers. Data we have gathered suggests that novice programmers do not easily acquire such a plan (Soloway et al. 1982, Soloway et al. 1983).

First of all, novices often want the WHILE to have a demon like structure. Consider, for example, the following transcript:

Subject: How do I get [the WHILE loop][2] to do that over again? See, I guess I don't know, I thought I had it. What happens now, how do I get it to go back? ... I say to myself, why would it do [the WHILE test] after [the last line of the loop body]? It seems to me that it would do it as soon as the [variable tested in the WHILE condition] changes. ...

Interviewer: So how will the WHILE statement behave?

Subject: Again, total guess here, I'm saying the WHILE statement, here's a logical guess ... everytime [the variable tested in the WHILE condition] is assigned a new value, the machine needs to check that value ...

The subject's "logical guess" is that the condition in the WHILE loop is being *continually* tested, and that the loop will be exited as soon as the condition is true. This is not an unreasonable interpretation; it is is consistent with the meaning of "while" in English phrases such as "while you are on the highway, watch for the Northfield sign". In a group study with novice programmers, we found that 34% had this type of misconception about the test in the WHILE loop (Soloway et al [1981a]).

Novices also try to implement the *get a value, process that value* plan, even though they are programming in Pascal. Consider, the following novice program fragment,

```
  ...
VAR Count, Total, I : INTEGER;
 BEGIN
 Count := 0
 Total := 0
 Writeln ('Enter integer')
 Read (I)
 WHILE I <> 99999 DO
   BEGIN
   Count := Count + 1
   Total := Total + I
   Read (I)  the subject has crossed out this
   END                line out after writing it down
  ...
```

and a transcript of the subject discussing this program:

Subject: If I put a number in [at the top of the loop], it comes through [the loop body]. I don't think I want [the inside Read] read again, I want it read up [at the top of the loop] ... If I read it [at the bottom of the loop body], what's that going to do for me? It's not going to do anything for me. OK, if I come out of the loop, having entered [a value], finish all [the loop body], then if I read in another one [points to Read above the WHILE, traces a flow from that outside Read down through the loop]. I guess what I need to figure out is how do I get back up here [points to the Read above the WHILE].

The subject wants to put the Read at the top of the loop, making the test in the middle of the loop. This reflects the *get a value, process that value* plan. In a separate study Soloway, et al. [1983] show that a new Pascal looping construct supporting this plan significantly improved novice and intermediate performance with Problem 2.

Conflicts and problems can occur even when the novice appears to fully understand a program fragment. Consider, for example, the following novice. She is writing pseudo-code for the following problem:

> ***Problem 3:*** Write a program which reads in 10 integers and prints the average of those integers.

After working on the problem for a few minutes, she had written the following:

```
Repeat
(1) Read a number (Num)
       (1a) Count := Count + 1
(2) Add the number to Sum
       (2a) Sum := Sum + Num
(3) until Count := 10
(4) Average := Sum div Num
(5) writeln ('average = ', Average)
```

Leaving aside some inconsistent pseudo-code notation, this is correct. At this point, the interviewer asks whether the statement on line 1a is the "same kind of statement" as that on line 2a. The subject seems to understands the role these two lines play in the program. She also recognizes the need for other associated statements to carry out those roles. Nonetheless, it appears that she thinks the Pascal translator knows far more about these roles than it does:

Interviewer: Steps 1a and 2a: are those the same kinds of statements?

Subject: How's that, are they the same kind. Ahhh, ummm, not exactly, because with this [1a] you are adding - you initialize it at zero and you're adding one to it [points to the right side of 1a], which is just a constant kind of thing.

Interviewer: Yes

Subject: [points to 2a] Sum, initialized to, uhh Sum to Sum plus Num, ahh - thats [points to left side of 2a] storing two values in one, two variables [points to Sum and Num on the right side of 2a]. Thats [now points to 1a] a counter, thats what keeps the whole loop under control. Whereas this thing [points to 2a], this was probably the most interesting thing ... about Pascal when I hit it. That you could have the same, you sorta have the same thing here [points to 1a], it was interesting that you could have, you could save space by having the Sum re-storing information on the left with two different things there [points to right side of 2a], so I

[2]Text in square brackets ("[" and "]") describes items pointed to by the subject. Usually the subject's actual words were "this", "here", or something similar. The brackets and words were used to make the transcripts more readable.

didn't need to have two. No, they're different to me.

Interviewer: So -- in summary, how do you think of 1a ?

Subject: I think of this [point to 1a] as just a constant, something that keeps the loop under control. And this [points to 2a] has something to do with something that you are gonna, that stores more kinds of information that you are going to take out of the loop with you.

This interview explains a result we have from an earlier written study. We found 100% of novices working on problems like 2 and 3 were able to correctly write the counter variable update statement ("`Count := Count + 1`"), while only 83% could correctly write the running-total variable update ("`Sum := Sum + Num`") [Soloway et al, 1982]. Why this difference with statements syntactically and semantically so similar? With this transcript, we now have some insight into the problem. Our subject seems to be keying on the role -- the *pragmatics* -- of the statements, noticing but not concentrating on the syntactic and semantic regularity. The running-total variable update is more difficult because it "stores information that you are going to take out of the loop with you". That is, it has implications outside the loop body.

4. Conclusions

The implication of these results is not simply to make syntactic fixes to programming languages. Instead, we are suggesting that the knowledge people bring from natural language has a key effect on their early programming efforts. Shneiderman and Mayer [1979] have proposed a model of programmer behavior based on language specific knowledge (which they call *syntactic*) and more general programming knowledge (called *semantic*). Our results suggest that there is a third body of *natural language step-by-step specification knowledge* which strongly influences novice programming behavior.

Miller [1981], Green [1981], and others have previously looked at step-by-step natural language specifications. They concentrated on looking at the suitability of natural language for directing computers. Based on the ambiguities and complexity limitations of natural language, they concluded it would be quite difficult to program in natural languages. Here, we are not contradicting that result, but extending it. We are finding that novice programmers *do* use natural language, even when they think they are using a programming language.

There are several implications of this work for programming education. First, we note that the power of the notions from structured programming will only be useful to students who have mastered the level of pragmatic and tacit programming knowledge highlighted in this paper. We need to address the problems students have very early in their programming education. The errors discussed here are barriers for many programming students. Only after a student has mastered writing a simple loop, for example, is he or she ready to see the power of a top-down design involving several loops.

We are beginning to explain many novice programming errors through the idea of natural language step-by-step specification plans. The quality of these explanations has proved important in the development of a tutor to do intelligent computer assisted instruction of programming [Soloway et al., 1981b]. In the future, we hope to extend the tutor to understand a stylized form of these natural language plans.

Though use of our plans cannot yet be fully automated, such plans can still play a part in a programming curriculum. As we stated earlier, the knowledge contained in such plans is usually tacit. Programming teachers, we feel, have much to gain by making that knowledge as explicit as possible as early as possible. We are presently developing an introductory course where students are taught both natural language step-by-step specification plans and programming plans from the beginning. Not only is the information in plans made explicit, but the differences between similar plans for different languages, in particular the natural language and the programming language being studied, can be made explicit. (These ideas are developed fully in Bonar [1983].)

Finally, what is the key to cognitively appropriate novice computing systems? Our work suggests that we need serious study of the knowledge novices bring to a computing system. For most computerized tasks there is some model that a novice will use in his or her first attempts. We need to understand when is it appropriate to appeal to this model, and how to move a novice to some more appropriate model.

5. References

Bonar, J., K. Ehrlich, E. Soloway, and E. Rubin, (1982) Collecting and Analyzing On-Line Protocols from Novice Programmers, in *Behavioral Research Methods and Instrumentation*, May 1982.

Bonar, J. (1983) Natural Problem Solving Strategies and Programming Language Constructs: Conflicts and Bridges. Ph.D thesis in preparation.

Collins, A. (1978) Explicating the Tacit Knowledge in Teaching and Learning, presented at the American Education Research Association (also Bolt Beranek and Newman Technical Report 3889).

DiSessa, A., (1982) Unlearning Aristotelian Physics: A Study of Knowledge-Based Learning, *Cognitive Science*, 6:1 (January-March), pp. 37-75.

Du Boulay, B. and T. O'Shea (1981) Teaching Novices Programming, in *Computing Skills and the User Interface* edited by M.J. Coombs and J.L. Alty, Academic Press, New York.

Green, T. (1981) Programming As a Cognitive Activity, in *Human Interaction With Computers*, edited by C. Smith and T. Green, Academic Press.

Miller, L. A. (1981) Natural language programming: Styles, strategies, and contrasts, *IBM Systems Journal*, 20:2, pp. 184-215.

Rissland, E. (1978) The Structure of Mathematical Knowledge. *Cognitive Science*, 2:4 (October-December 1978).

Shneiderman, B. and R. Mayer (1979) Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results, *International Journal of Computer and Information Science*, 8:3, pp. 219-238.

Soloway, E., J. Bonar, B. Woolf, P. Barth, E. Rubin, and K. Ehrlich (1981a) Cognition and Programming: Why Your Students Write Those Crazy Programs, appeared in Proceedings of the National Educational Computing Conference, pp. 206-219.

Soloway, E., B. Woolf, E. Rubin, J. Bonar, W. L. Johnson, (1983) MENO-II: An Intelligent Programming Tutor, *Journal of Computer-Based Instruction*, in press.

Soloway, E., K. Ehrlich, J. Bonar. J. Greenspan, (1982) What Do Novices Know About Programming?, in *Directions in Human-Computer Interactions*, edited by B. Shneiderman and A. Badre, Ablex Publishing Company.

Soloway, E., J. Bonar, and K. Ehrlich (1983) Cognitive Factors in Looping Constructs, *Communications of the ACM*, to appear.

Waters, R. C., (1979) A Method for Analyzing Loop Programs, *IEEE Transactions on Software Engineering*, SE-5:3, May.

PREDICTING STUDENT SUCCESS IN AN INTRODUCTORY PROGRAMMING COURSE

by Terry R. Hostetler

Global Analytics, Inc.
10065 Old Grove Road
San Diego, California 92131

Abstract

This paper examines to what extent a student's aptitude in computer programming may be predicted through measuring certain cognitive skills, personality traits and past academic achievement. The primary purpose of this study was to build a practical and reliable model for predicting success in programming, with hopes of better counseling students. Results from correlating predictor variables with a student's final numerical score confirmed past studies which showed the diagramming and reasoning tests of the Computer Programmer Aptitude Battery and a student's GPA to be the predictors most closely associated with success. A multiple regression equation developed from 5 predictors correctly classified 61 of 79 students (77.2%) into low and high aptitude groups.

1 Introduction

Instruction in computer science has become increasingly desirable due to several expanding areas of demand. Many academic departments have realized the importance of computer assistance to their discipline, and so have adopted computer science requirements. The public has recognized a growing practical need to be computer literate in order to function as intelligent consumers and take advantage of today's technology. Employers have begun scrutinizing potential applicants for computer training, causing students to carefully consider as extensive a computer science background as possible. These and other demands have resulted in a recent saturation of computer science courses, including the traditional first course, introductory programming.

Advising students who are making their initial contact with computer science in curriculum decisions has always been difficult. The increased volume of students, however, has made it of even greater importance to devise a useful method for determining individualized counseling. Both students and computer science departments can benefit greatly from a screening technique in which students can compare their interest in programming to their projected aptitude.

At educational institutions, such as the University of Illinois at Urbana-Champaign, where the student population has in general met uniformly high admission's criteria, it might be thought that these students would perform at a consistently high level in an introductory programming course. Those involved in teaching such courses, however, observe that student ability has a remarkably broad range, even within apparently homogeneous groups in a particular college.

Considering this seemingly extensive range of aptitude, it becomes desirable to question whether it is possible to identify traits in an individual which may be used to predict how successful that person will be in an introductory programming course. This paper investigates to what extent certain cognitive skills, personality variables, and past academic achievement can be used to develop such a predictive scheme.

2 Design

2.1 Sample Group

Students used in this study were enrolled in Computer Science 105 at the University of Illinois at Urbana-Champaign during the spring semester of 1982. Computer Science 105, entitled "Introduction to Computers and Their Application to Business and Commerce", is offered every semester and has no prerequisite.

Computer Science 105 covers the basic concepts of structured programming, using an extended version of Fortran supported by the Watfiv compiler. Its major topics include: organization of the computer, data types, variables, arithmetic expressions, assignment statements, input/output, control flow, multidimensional arrays, subprograms, and sorting and searching lists.

Students are evaluated according to their performance on programming assignments (seven to nine), two one-hour examinations, and a three-hour final examination. Programming assignments require students to apply concepts from lecture material to solve stated problems, using an IBM 4341 computer in time-sharing mode. Exam material ranges from objective questions on theory and facts to the actual writing of programs.

A sample of 120 students was randomly selected from approximately 600 students enrolled. This sample was reduced by 17 students who withdrew before the end of the semester. Missing data forced the elimination of another 24 students, leaving a final sample size of 79. This final sample consisted of: 35 males and 44 females; 34 freshmen, 14 sophomores, 18 juniors, 12 seniors, and one graduate student.

2.2 Selection of Data Variables

Two tests from the Computer Programmer Aptitude Battery (CPAB) were chosen as measures for this study. The author, Palormo [Palo74], describes these tests as:

Reasoning--a test of ability to translate ideas and operations from word problems into mathematical notations.

Diagramming--a test of ability to analyze a problem and order the steps for solution in a logical sequence.

Palormo reports estimated reliabilities of 0.88 for the reasoning test and 0.94 for the diagramming test. Validity results show correlations with training success over four studies varying from 0.43 to 0.52 for the reasoning test and from 0.25 to 0.69 for the diagramming test. These two tests recorded the most consistently high correlations of the five tests composing the battery.

In the only other study located in which the CPAB was included as a predictor, Mussio and Wahlstrom [MuWa71] found the diagramming test of the CPAB the single best predictor of course grade, supporting their conclusion that reasoning ability is the single most important qualification for programmers. They feel that the diagramming test closely resembles basic demands that are relevant to programming, and that it appears the logic required to solve the test questions is also important and necessary in a training situation. Similarly, Johnson [John72], in his review of the battery, asserts that the reasoning test of the CPAB represents a task very close to that of programming.

In the area of personality, Weinberg [Wein71] states that there appears to be evidence indicating that critical personality factors can be located and associated with particular tasks, at least to the extent that their possession may render one incapable of performing that task well. Alspaugh [Alsp72] found that the more successful programming student might be expected to have a personality associated with a low level of "impulsiveness" and "sociability" and a relatively high level of "reflectiveness" as measured by the Thurston Temperament Schedule.

I selected Form A of the Sixteen Personality Factor Questionnaire (16PF) to be administered for this study. Its manual [IPAT79] describes the 16PF as an objectively scored test constructed through basic research in psychology to provide the most complete coverage of personality possible in a brief time. This comprehensive coverage is based on the measurement of 16 functionally independent and psychologically meaningful traits (PF01, PF02, ..., PF16).

The manual reports the overall reliability of factor scores as quite good, even over a four-year period. Validity coefficients are shown to be exceptionally high, meaning the test questions are good measures of personality traits, as these traits are represented in research analysis.

The significance of past academic achievement in predicting programming success has been established for certain variables by several studies. Petersen and Howe [PeHo79], Fowler and Glorfeld [FoGl81], and Bauer, Mehrens, and Vinsonhaler [BaMV68] all reported college grade point average (GPA) as the single best predictor of success in the models they developed. Studies both by Fowler and Glorfeld [FoGl81] and Alspaugh [Alsp72] found a student's mathematical background to be an important contributing element in estimating success.

GPA and math background were both included as variables in my study. Student GPA's were collected from the registrar as of the beginning of the spring semester 1982. A student's math background was measured according to a scale similar to one used in Alspaugh's [Alsp72] study. This scale associates an integer with the most advanced mathematics course a student has passed, with a higher integer indicating a more advanced course.

The primary criterion of success in Computer Science 105 was chosen to be final numerical score. A student's final numerical score is computed as a weighted sum of objectively graded programming assignments and exams. Students are ranked according to these scores, and then final grades are assigned. In the judgment of those teaching the course, the final numerical score is the most consistent and accurate measure available of successful performance.

The cognitive and personality tests were administered during the first two weeks of the semester in two one-hour sessions. Session one included Form A of the 16PF, an untimed test requiring approximately 40 minutes. The reasoning and diagramming tests of the CPAB, with corresponding test times of 20 and 35 minutes, were given in session two. Results of the tests were not made available to the students.

A summary of the 21 independent and dependent variables used in this study, along with their assigned abbreviations and sources, is displayed in Table 1.

Table 1

Data Variables

Variable	Abbreviation	Source
Cognitive:		
CPAB		
—Reasoning	REASON	CPAB
—Diagramming	DIAGR	CPAB
Personality:		
16PF		
—Reserved/Warmhearted	PF01	16PF
—Less Intelligent/ More Intelligent	PF02	16PF
—Affected by Feelings/ Emotionally Stable	PF03	16PF
—Humble/Assertive	PF04	16PF
—Sober/Happy-go-lucky	PF05	16PF
—Expedient/Conscientious	PF06	16PF
—Shy/Venturesome	PF07	16PF
—Tough-minded/Tender-minded	PF08	16PF
—Trusting/Suspicious	PF09	16PF
—Practical/Imaginative	PF10	16PF
—Forthright/Shrewd	PF11	16PF
—Unperturbed/Apprehensive	PF12	16PF
—Conservative/Experimenting	PF13	16PF
—Group Oriented/ Self-sufficient	PF14	16PF
—Undisciplined Self-conflict/ Controlled	PF15	16PF
—Relaxed/Tense	PF16	16PF
Academic:		
College GPA	GPA	Registrar
Math Background	MATH	Questionnaire
Success:		
Final Numerical Score	SCORE	Instructor

3 Results

The sample of students was randomly divided into two groups: Group A (64 students), used in performing a bivariate correlation analysis and in developing a multiple regression equation, and Group B (15 students), used to cross-validate this regression equation.

Pearson product-moment correlation coefficients were generated for Group A to measure the degree to which variation in each independent variable relates to variation in the dependent variable. Correlations were also tested for significance from zero. Correlations for all predictor variables and their associated significance levels are presented in Table 2.

The predictor most highly associated with success (SCORE) was found to be the diagramming test of the CPAB (DIAGR), followed in order by the reasoning test of the CPAB (REASON) and a student's GPA. These measures obtained highly significant correlations of 0.480, 0.406, and 0.367 respectively. No other independent variables were found to significantly correlate with a student's final numerical score.

Multiple regression using stepwise inclusion was performed with all the independent variables in this study. With stepwise inclusion, the variable that accounts for the largest amount of variance unexplained by the variables already in the equation, enters the equation at each step. The results of this analysis are summarized in Table 3.

Table 2

Correlations of Independent Variables with Final Numerical Score (SCORE) (N = 64)

Independent Variable	Correlation Coefficient
REASON	.406**
DIAGR	.480**
PF01	-.035
PF02	.145
PF03	-.121
PF04	-.056
PF05	-.059
PF06	-.088
PF07	-.025
PF08	.126
PF09	.072
PF10	-.068
PF11	.016
PF12	.008
PF13	-.17?
PF14	.036
PF15	-.043
PF16	.102
GPA	.367**
MATH	.030

** p<.01

Table 3

Stepwise Multiple Regression Analysis (N = 64)

Step	Variable Entered	Multiple R	Simple R
1	DIAGR	.480**	.480**
2	GPA	.568**	.367**
3	REASON	.608**	.406**
4	MATH	.632**	.030
5	PF05	.653**	-.059
6	PF08	.658**	.126
7	PF10	.666**	-.068
8	PF09	.671**	.072
9	PF01	.677**	-.035

** p<.01

The multiple correlation (multiple R) considering only the best predictor (step 1), DIAGR was increased from 0.480 to 0.568 with the addition of GPA (step 2). This multiple R was further improved to 0.608 by including REASON (step 3). Although MATH was not significantly correlated with the success criterion, its addition to the model (step 4) raised the multiple R to 0.632. Similarly, PF5, the first personality factor added (step 5), improved the multiple R to 0.653, even though it did not directly correlate with success. The addition of the remaining variables to the regression equation resulted in minimal increments to the multiple R.

Table 4 provides the beta weights and R squared for this five-variable model. Beta weights, the standardized regression weights, show the relative contribution of the corresponding predictor variables to the success criterion. The R squared indicates the proportion of variation in the criterion measure explained by the predictors.

Table 4

Statistics for
Five-Variable Regression Model
(N = 64)

Independent Variable	Regression Coefficient	Beta Weights
DIAGR	.593	.385
GPA	6.442	.330
REASON	.663	.260
MATH	2.260	.191
PF05	-.782	-.167
(constant)	21.675	

Multiple correlation = 0.653**; R squared = 0.427

** p<0.01

The five-variable model was cross-validated using the 15 students in Group B. The cross-validation correlation between the predicted final numerical scores and the actual scores was found to be 0.672 ($p<0.01$). This cross-validation R suggests that the relationships identified by the model hold true for different samples drawn from the same population. In other words, the formulated model does not appear to be sample dependent.

All analysis described in this study was performed using the Statistical Package for the Social Sciences [NieN75].

4 Discussion and Recommendations

The multiple R of 0.653 obtained for the five-variable equation developed in this study is comparable to research by Alspaugh [Alsp72] and Mussio and Wahlstrom [MuWa71] whose models, measuring a similar combination of traits, derived multiple R's of 0.632 and 0.67 respectively. This study's model explained approximately 43% of the variance in the final numerical scores of students, leaving a majority of the variance unaccounted for.

As with Mussio and Wahlstrom's work [MuWa71], the diagramming and reasoning tests of the CPAB were found to be the highest correlating independent variables (0.480 and 0.406). DIAGR and REASON, when summed together, correlated 0.533 ($p<0.01$) with success in the course. These results support the contention that reasoning is a cognitive skill important to programming.

None of the personality traits measured correlated significantly with success in the course (SCORE). PF05, though, as the fifth variable added in the multiple regression analysis, improved the multiple R from 0.632 to 0.658.

A student's GPA was found to correlate highly significantly with SCORE, supporting past research which reflects the tendency of past academic success to be a good predictor of current academic success. MATH, the fourth variable added during the multiple regression, raised the multiple R from 0.608 to 0.632, despite its low correlation with success (0.030).

The primary goal of this study was to develop a practical and reliable model for predicting success in an introductory programming course, with the major benefit of being able to better counsel students in curriculum decisions.

In view of this goal, Table 5 shows a breakdown of the student sample according to high and low aptitude, based on predicted and actual final numerical scores. High aptitude is defined to be a final numerical score associated with a final letter grade of an A or B, while low aptitude is designated as a score resulting in a C, D, or E.

Table 5

Breakdown of Student Aptitude
for Five-Variable Model

		Actual: Low	Actual: High	Total
Predicted	Low	30	10	40
	High	8	31	39
	Total	38	41	79

The table shows that 30 out of 40 students (75%) who were predicted to have low aptitude, and 31 out of 39 students (79.5%) who were predicted to have high aptitude, actually did attain those levels of aptitude. Overall, the model correctly classified 61 out of 79 students (77.2%) into low and high aptitude.

Prediction of success based on the models presented in this study is a useful technique in counseling students. A majority of the variance in the success criterion was not explained, however, requiring the consideration of other factors not measured that will aid in predicting a student's success, such as measures of an individual's desire and motivation. Future research in this area must be done in an attempt to identify such traits.

It is important to note that this study dealt with a highly homogeneous, pre-selected group of students. Most members of the sample group were enrolled in the College of Commerce and Business Administration, for which CS105 is a required course. This college reports that its students entering in 1981 had an average ACT composite score of 27 and high school class rank of 92%.

Acknowledgements

I am grateful to Kikumi and Maurice Tatsuoka, Geneva G. Belford and Gerard M. Chevalez for their help throughout this research. I also extend appreciation to the students of my sample group for participating in this study.

References

[Alsp72] Alspaugh, Carol Ann. "Identification of Some Components of Computer Programming Aptitude." Journal for Research in Mathematics Education, March 1972, pp. 89-98.

[BaMV68] Bauer, Roger, William A. Mehrens, and John F. Vinsonhaler. "Predicting Performance in a Computer Programming Course." Educational and Psychological Measurement, 28 (1968), pp. 1159-64.

[FoGl81] Fowler, George C. and Louis W. Glorfeld. "Predicting Aptitude in Introductory Computing: A Classification Model." AEDS Journal, Winter 1981, pp. 96-109.

[IPAT79] Institute for Personality and Ability Testing. Administrator's Manual for the 16 PF. Champaign, Ill: IPAT, 1979.

[John72] Johnson, Richard T. Rev. of the Computer Programmer Aptitude Battery. In the Seventh Mental Measurements Yearbook. Highland Park, N. J.: Gryphon Press, 1972.

[MuWa71] Mussio, Jerry J. and Merlin W. Wahlstrom. "Predicting Performance of Programmer Trainees in a Post-High School Setting." Proceedings of the Annual Computer Personnel Research Conference, 1971, pp. 26-45.

[NieN75] Nie, Norman H., et al. Statistical Package for the Social Sciences. 2nd ed. New York: McGraw-Hill, 1975.

[Palo74] Palormo, Jean M. Computer Programmer Aptitude Battery: Examiner's Manual. 2nd ed. Chicago: Science Research Associates, 1974.

[PeHo79] Petersen, Charles G. and Trevor G. Howe. "Predicting Academic Success in Introduction to Computers." AEDS Journal, Fall 1979, pp. 182-91.

[Wein71] Weinberg, Gerald. The Psychology of Computer Programming. New York: Van Nostrand Reinhold, 1971.

COMPUTER ASSISTED INSTRUCTION

Michael G. Southwell
Mary Epes
J. Kenneth Sieben
Ellen Leahy
R. K. Wiersba

ABSTRACT: Computer Assisted Sentence Combining

Michael G. Southwell, Carolyn Kirkpatrick, Mary Epes, Department of English, York College/CUNY, Jamaica, NY 11451

We have reported at previous meetings of NECC on The COMP-LAB Writing Modules, a set of computer-assisted grammar lessons being developed at York College. Here we want to demonstrate the latest development, a set of lessons to foster students' knowledge of, and ability to use, the complex sentence patterns of written English.

Many college students who have had limited writing practice experience difficulty moving beyond simple sentence patterns in their writing. Sentence-combining exercises have become popular among high school and college teachers, because research has suggested this to be one of the few pedagogies that really can change the way students write. But such exercises are time consuming and unwieldy as whole-class activities, and are too complicated to assign as unsupervised homework. However, computers afford a way to build such activities into the curriculum.

In designing our materials, we have sought to exploit the ways in which computers can provide instruction which is more effective than that possible with print or even video materials. In particular, we use the computer's capacity for dynamic presentation to help students understand the structural components of sentences, then we tap its interactivity to help them practice manipulating these components.

Understanding. As we shall demonstrate, computers can highlight the distinction between the clauses of sentences, and the connecting words which tie them together; and they can connect the clauses in different ways to show different logical relationships. Then they can demand that students make responses that demonstrate their understanding.

Practice. Students must have some opportunities to practice manipulating the parts of a sentence, actively creating new sentences by combining short sentences with single ideas into complex sentences with more complicated meanings. As we shall demonstrate, computers make it possible to separate the cognitve work and the mechanical work: students tell the computer where and how to combine ideas, and then the computer does the actual combining. This makes it feasible to provide students with those large quantities of practice which are necessary to have a real impact on students' writing.

ABSTRACT: How to Write Computer Assisted Instructional Programs to Support a Textbook

J. Kenneth Sieben, 86 River's Edge Drive, Little Silver, NJ 07739

The author is in the process of developing CAI lessons for each of the exercises in his textbook Composition Five (co-authored with Dr. Lillian Anthony). The goal of the project is to remove the time-consuming, and often boring, task of homework correction from the classroom.

We teach a lesson on a reading or writing skill and assign exercises from the text to be done as homework. However, instead of having to review those exercises item by item, we can send our students to the Computer Lab to correct their own. The student keys in his responses, with letters (T-F), (a, b, c, d), words, or even entire sentences for the many sentence combining exercises. The computer is programmed to respond to a variety of possible answers in such a way as to teach the student whatever he/she did not understand. Students are given the instruction to check those items for which the computer did not give a satisfactory explanation and see us during office hours for individual help. This procedure frees up classroom time for development of cognitive skills which (we think) can best be done in an atmosphere of free exchange. It also provides for complete individualization of homework evaluation.

The fact that we wrote our own textbook might have made it easier to develop support CAI materials. However, I believe that any teacher could learn to program almost any kind of exercise to enable students to correct their own work at their own pace. I will be happy to share with colleagues some of the problems and solutions regarding the development of CAI materials.

ABSTRACT: Project Better Chance - A Comprehensive Approach to Basic Skill Improvement: A Better Chance for High-Risk Students - (A Title III Project) CAI Component for Reading and Writing Skills Reinforcement

Ellen Leahy, Bronx Community College, Sage Learning Center, W. 181 St. & University Ave., Bronx, NY 10453

Setting. A special block program was offered to first year students scoring below acceptable levels on the CUNY Assessment Exams. One component of this project is computer assisted instruction with software developed in collaboration with the Reading and English faculty who are on the Basic Skills team and the CAI Specialst.

Software. One program, "Dictionary", consists of a series of practice exercises designed to reinforce skills developed with the instructor in class. Word meanings, etymology, and words with irregular plural spelling are examples of the content of the program. Students must type in their answers (words or phrases) to questions. This requirement was included to reinforce spelling skills in context. Students receive immediate feedback and a printout of their incorrect responses so that they leave the Computer Lab with individualized study material. Their instructor also receives a copy of the printout for each student completing the practrice exercises.

Another program is a series of commonly used vocabulary development review tests in three levels of difficulty with a total of 14 tests on the diskette. This program was designed to remove a regularly scheduled test exercise from the class period. The transfer of this activity to the Computer Lab allows students to set their own pace for taking the test, schedule themselves for the test at a time most convenient for them, and more importantly, gives the instructor more class time to develop other skills. This program provides immediate feedback to the student on test performance with a list of incorrect answers and the test score. At the end of the test deadline, the instructor receives copies of the student printouts.

Both of these programs have a management system which generates class performance reports to the instructors. The programs are written in MBASIC, requiring a Z-80 card for the Apple II+ with 48k.

Another program is a series of 3 practice tests for a Health Education course on the topics of Mental Health, Drugs, and Human Sexuality. Another program covers the same topics but has restricted use as unit make-up exams. The following commercial programs are being tried out with individual students: Compupoem, MECC English, MECC Micro-quest, and Micro-vocab for spelling practice exercises. In the process of development are programs for reinforcing Reading for Main Idea, Outlining, and Subject-Verb Identification and Word Endings for ESL students.

Facility. The Computer Lab is located in the Sage Learning Center of Bronx Community College, and houses 10 Bell and Howell Apple II+ microcomputers. Epson MX-80 printers produce hard copies of individual student reports and the student data management system reports.

ABSTRACT: Appropriate Technology for Computer Education

R. K. Wiersba, Bentley College, Waltham, MA

The successful application of computers to virtually all aspects of our culture leads logically to two statements relating to computer education:

1) Qualified instructors in computer topics will be in short supply. As the ability of a healthy computer industry to offer a higher financial return for computer expertise increases, qualified computer instructors will gravitate to the industrial sector. This is already occurring.

2) At a time when instructional resources will be strained, a greater proportion of college students will enroll in various computer courses. Although it appears there will be a decreasing number of total college enrollments, curriculum changes are being made in various academic departments to reflect the need for computer literacy, causing an increase in the number of students enrolled in computer course as a proportion of total enrolled students. This is most eveident in courses offered at the introductory level.

In an attempt to upgrade computer education in general and to contribute to helping instructors perform at a higher level, the author encourages computer educators to modernize their taching methods. This will also provide an example for other disciplines to follow in the use of the computer in the classroom. This can be done and is being done with self-contained instructional modules which can be designed to illustrate the operation of specific logical computer mechanisms in depth, under the control of the instructor or the student.

The author has studied the computer programming topics taught at his institution which have lent themselves most readily to blackboard illustration, and has developed, with student programming help, a pilot CAI module which illustrates the operation of linked lists and binary trees, and a second for the COBOL MOVE command. These are used in classrooms equipped with instructors' terminals and multiple television monitors for students. The most immediate advantage accruing from the use of these modules is their repeatability, both by the instructor in the classroom and by the students outside of classes, either on campus or by dial-up lines. Their operation is menu-driven and requires no additional explanation or documentation.

The amount of programming required to complete these modules appears great, considering the amount of time involved in their actual execution. However, as a bonus, the design and construction of these modules (written in COBOL, under the author's supervision), proved a valuable project experience for the students involved. Control of a variety of different terminal cursors was one of the significant challenges and led to a design decision in favor of table-driven screens for ease of modification to accomodate different vendors' control code schemes.

Since completing the first phase of this project, the author has become aware of at least one other group of college instructors involved in developing a similar instructional module. Communication is solicited from others interested in these efforts, with in mind exchanging advice and design information and eventually, the modules themselves.

COMPUTERS IN EDUCATION AT AN EARLY EDUCATION LEVEL

Carol L. Clark
Elizabeth Legenhausen
Stewart A. Denenberg
Marilyn J. Pollock

ABSTRACT: The Magic Crayon - An Introductory Computer Experience for Children

Carol L. Clark, 5713 Kentford Circle, Wichita, KS 67220

Children love to make things happen. They also love computer graphics, especially colorful, dynamic ones. Magic Crayon is a program designed to capitalize on these interests and provide a positive introductory computer experience for childeren.

The program permits children to draw on the Apple II's low resolution graphics screen, using keystrokes to control direction and select colors. They can instruct the computer to "remember" a picture, and then command it to dynamically reproduce the drawing at a later time.

For children too young to read, special pictorial instructions are provided; the program has been used successfully by children as young as three years old. For older children, more advanced options are available, including changing a drawing's location on the screen and combining several picture elements into one large drawing. This provides an exciting introduction to some of the most fundamental concepts of computer logic.

An important aspect of Magic Crayon is the "feeling of power" given to the child. Children feel that they are in control of the machine as it responds to their commands. Even the youngest children can discover that computers do exactly as they are instructed, in response to decisions made by the people who control them.

Plans for further field testing in classrooms and homes are being made. In addition to a demonstration of Magic Crayon's capabilities, this project presentation will relate results of these field experiences, including indication of the program's applicability for various age groups and disciplines.

ABSTRACT: Effectiveness of Computer Usage on Achievement of Specific Readiness Skill of Preschoolers

Elizabeth Legenhausen, 6517 Beverly Road, Baltimore, MD 21239

The purpose of this non-randomized, experimental, pretest-posttest research design was to determine the effectiveness of computer usage on the achievement of specific readiness skills in preschool children. In addition, observational data was collected regarding the ability of preschoolers to master computer operation.

Subjects selected for this study were 30 four year old, upper middle class children, 16 boys and 14 girls, registered in a private nursery school program in Baltimore, Maryland. An experimental group and a control group were chosen at random from two preassembled classes.

During the six week experimental program, the control group received the traditional preschool curriculum. (Ethical considerations dictated that subjects in the control group be allowed equal time using the microcomputer.) Concepts chosen as targets for instruction and measurement in the experimental program were classified into three context categories: Space, Quantity, and Time. Subjects in the experimental group received two group introductory sessions about microcomputer usage. Then, pairs of children were allowed to select an Apple II Plus microcomputer for 10 minute periods during free play time each morning. The computer was equipped with two commercially produced programs: Juggle's Rainbow, produced in 19 The Learning Company; and Count Bee, produced in 19 by Edu-Ware, Inc.

The Boehm Test of Basic Concepts selected to collect data for this study. Form A and Form B were used as pretest and posttest, respectively. Statistical analysis of raw scores on the pretest and posttest for the experimental and control groups was implemented in three steps. First, the percentage of correct responses in each context category on the pretest and posttest for the experimental and control groups was calculated. Second, mean and standard deviation of scores in each category on the pretest and posttest for the two groups were computed. Third, t-tests were used to determine whether differences between mean scores for the experimental and control groups on the pretest and posttest were significant at the .05 level.

The major finding of this study indicated that microcomputer usage at the preschool level was statistically significant at the .05 level. In addition, preschool children in this research study enthusiastically mastered computer instruction!

ABSTRACT: The Oak Street Interns: An Experiment

Stewart A. Denenberg, Associate Professor of Computer Science and Director of Academic Computing, SUNY Plattsburgh, Plattsburgh, NY 12901

During the Fall semester of 1982, eight undergraduate computer science interns taught third graders how to program in BASIC using Apple microcomputers at the Oak Street Elementary School in Plattsburgh, New York. A particular pedagogy and curriculum has been developed which focuses on the graphic capabilities of Apple BASIC and, as a result, is not presented in the "normal" order of most textbooks. Briefly, the following curriculum plan is used (all lessons after the first are begun with a review of the previous lesson):

(1) Familiarize child with the keyboard, diskettes, and disk drive. Play educational games: Lemonade, Hangman, Speedreading. Learn to Boot, CATALOG, RUN, and how to correct typing errors. Use of RETURN key and CTRL-C. Use keyboard visual aid.
(2) Child runs Intern-written program that accepts a color number, "how far to go over" and "how far to go down", then plots that point in low resolution graphics. After child has mastered the coordinate system, they use the program to draw (in increasing difficulty) Plus (+), Times (x), diagonal (/), square, rectangle, right triangle, isoceles triangle.
(3) Child now motivated to write their own plotter program so GR, TEXT, COLOR = , and PLOT are taught in the Immediate mode (student can always see last plot command in lower window). Make some of the figures in lesson two above. Use numbered graph paper as visual aid.
(4) Program is lost in Immediate mode so motivation is provided to teach how program can be saved using line numbers for each command. Experiment by changing colors. New commands: NEW, LOAD, SAVE. Homework: draw a picture on graph paper.
(5) Form pairs. Child A & B swap pictures from lesson four, each writes the program to create that picture. Intern gives children C & D the programs written by A & B, and C & D attempt to draw the pictures. Pictures compared.
(6) GOSUB. Show need for GOSUB when code is repeated. Use it to plot a 5x5 square. Show how to use GOSUB for modularity: draw a square as a set of horizontal lines, or a set of vertical lines using HLIN and VLIN.
(7) FOR-NEXT. Draw same square using a single loop then a double loop.
(8) Parents visit on last lesson. (Note: PRINT, INPUT, LET, and IF-THEN have purposely not been taught.)

Tentative results and conclusions reached thus far include:

- Because graphics is as close to a culture-free symbol system (unlike numbers or strings) it is easily understood and assimilated by young children.
- The absolute coordinate system in Apple BASIC presents no problems and it is unclear whether the relative addressing of LOGO is superior.
- Dedicated Computer Science Interns seem to thrive on this teaching experience, excelling at it even though they have no formal teacher training. Their presence has not only inspired the teachers at the school to learn more about microcomputers, the interns are exemplars of the public service function that a college can provide to the local school system.
- It may not be possible to develop a texbook that rigidly follows the proposed curriculum - many of the interns jumped forward and backward over topics and developed new examples on-the-fly in response to the individual needs of their learners. This is knowledge that accrues from having teachers well-steeped in the content knowledge and skills to be taught.
- The initial success of the project leads us to believe that it can be done about ten times more cheaply using the Timex Sinclair microcomputers. In the Spring of 1983, we will pilot test one group on a Sinclair using the same curriculum and pedagogy described above; what remains to be seen is if the multi-function keyboard presents problems to young minds (with, however, small fingers).

ABSTRACT: Why Computer Education in the Elementary School? A Model for Maximum Use

Mrs. Marilyn J. Pollock, 2 S. 621-F Fermi Court, Warrenville, IL 60555

A Philosophy for Maximum Use

I. Ideas from computer science and technology can expand an individual's learning strategies.
II. Learning to use a computer can help to achieve academic, personal, and career goals.
III. Learning to evaluate the purpose, values, and limitations of computer technology can enable an individual to make effective use of this technology in daily living.
IV. Learning to evaluate computer applications in terms of purposes, values, assumptions, and limitations will lead the student to analyze the possible social and political effects of various applications.

A Model for Maximum Use

Suggested beginning program in an individual school would address these questions:

1. Staff training
2. Communication to staff and parents
3. Software needs
4. Software maintenance
5. Scheduling
6. Long range plans

One approach to the above questions will be presented. Lesson suggestions will be available.

COMPUTER EDUCATION FOR SECONDARY SCHOOL TEACHERS

Susan M. Zgliczynski
Harriet G. Taylor
Dale M. Johnson
Carla J. Thompson
Dr. Sandra Crowther
Michel Eltschinger
Linda Hyler

ABSTRACT: Infusion of Microcomputer Training into the Existing School of Education Undergraduate and Graduate Curriculum

Susan M. Zgliczynski, School of Education, University of San Diego, Alcala Park, San Diego, CA 92110

Microcomputer use in educational settings is rapidly expanding. There is an ever-increasing need for educators - both teachers and administrators - to receive comprehensive training in the use of microcomputers in educational settings. Federal, state, and local funding to support inservice of practicing educators attempts to meet the needs of teachers and administrators in the elementary, middle, and secondary schools. New graduates of Schools and Colleges of Education are expected to have some training in the use of microcomputers in the classroom. In a tight placement market, new graduates with comprehensive training have a real advantage.

Many graduate and undergraduate programs for training educators have had difficulty instituting the necessary training. Lack of trained faculty, limited funding for equipment, and cutbacks on scheduling make it very difficult to provide comprehensive training. Most programs have attempted to meet the computer literacy needs of their students by adding an elective course on "The Use of Microcomputers in Education".

At the University of San Diego School of Education, the faculty saw several disadvantages to the above approach. First of all, our students had little room in programs leading to teaching, counseling, or administrative credentials to add an additional course. Tuition for an extra three-credit course in a student's program was an expensive burden at a private institution. The faculty also felt we weren't setting an example of integrating computer use into our curriculum while we were telling teachers and local school administrators that they should integerate computer use into existing elementary and secondary curricula.

A Faculty Research Grant was awarded to design modules to be fitted into existing undergratuate and graduate classes. A target goal was that within one year every student graduating with an undergraduate, masters, or doctoral degree would have received training in the operation of a microcomputer and most would receive training in computer applications in their area of expertise. Under the grant, one faculty member prepared the modules, inserviced the faculty, and presented the modules with the assistance of instructors in the classes.

Modules developed were as follows:

1. Regular Teaching Credential Program - Students in their first education class learned operation of the microcomputer and were taught ways the computer could be used in the classroom. Cautions about inappropriate use were included. Students in teaching methods classes were taught selection, evaluation, and use of CAI programs in their teaching fields.
2. Required Special Education Class - All students earning credentials complete this course. Students were introduced to the use of computer hardware and software used with the handicapped and got hands-on training working with handicapped students.
3. Counseling Credential Course - Students were introduced to the operation of the microcomputer, use of career guidance software, test interpretation and test preparation programs, use of word processing in producing reports, and data management and filing of student records.
4. Administrative Credential Course - Students were taught basic operation, selection and evaluation of hardware and software, preparation of a school needs assessment, and hands-on training in the use of administrative packages.
5. Doctoral Program in Educational Leadership - Students learned operation and BASIC programming, use of statistical packages, word processing, and use of modelling packages such as VisiCalc. Most of the students completed a research paper related to issues of computer use in their intended work setting. Several students now plan dissertation research related to microcomputer use.

During the first year each module was limited to 2-3 hours of classroom instruction, and a required application project for each course with a module. Six modules were developed during the initial year. Many students completed additional computer practice, attended workshops, and visited schools with model projects.

Faculty members have responded favorably to this approach. Many have obtained further computer training on their own. Most of them feel they can present the module to their classes with little assistance from the designer of the module in future seminars.

ABSTRACT: Certification of High School Computer Science Teachers

Harriet G. Taylor, Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803

The use of the computer as a part of the instructional process in the elementary and secondary schools in the United States is increasing at a rapid rate. A majority of this nation's high schools now have some facilities for instructional computing. Parents, teachers, and administrators are now insisting that schools prepare students to be a part of the computerized society in which they will live. As a result, computer science classes will soon be offered by most high schools.

Despite this growing area of national concern, few states have adopted certification standards for computer science teachers. It stands to reason that if computer competency is to be as essential a life skill as reading, writing, or arithmetic, then it is just as important to have standards for the teachers of computing.

Recently, a study was conducted to determine the national status of computer science certification. The major focus of this presentation will be a report on the results of the study. Included will be a summary of which states now offer certification in computer science and a composite description of those standards. These results will be compared to those obtained from a survey of leaders in the field of computer science education about the importance of certification and the content of certification programs. The presentation will conclude with questions from the audience and a general discussion of the issue of certification.

ABSTRACT: Introduction to Computers and Educational Computing - A CAI Approach

Dale M. Johnson, Research & Evaluation, University of Tulsa, Carla J. Thompson, Mathematics Department, Tulsa Junior College, Tulsa, OK 74104

The presentation will describe the development and content of a six unit CAI package consisting of 30 lessons (5 lessons per unit). Although numerous books, articles, and other print materials are available for introducing non-technical professionals to computing, there was a lack of computer instructional software on the very topic (educational computing) advocated by the print and A-V materials. Thus, the present project is an attempt to use the computer (as opposed to more traditional media) to teach about educational computing.

The units incorporated into the series are: (1) Hardware, Software, & People, (2) Program Development, (3) Internal Computer Functions, (4) Computers in Education, (5) Computing Issues in Schools, and (6) Selection and Evaluation of Hardware & Software. Programming as a topic was intentionally omitted because many existing computing CAI packages deal almost exclusively (and exhaustively) with programming in BASIC or some other high level language. Each of the 30 lessons are accompanied by one or more off-line activities designed to reinforce the CAI material.

The entire package was created using a special authoring system on the TRS-80 Model III computer system. Pedagogically, the system is tutorial in nature with terminal lessons in each unit designed for assessment purposes. Part I (Units 4, 5, & 6) is aimed at preservice and inservice educational personnel who are interested in computer literacy. The system has been designed, coded, and has completed technological debugging although field test evaluation with regard to its effectiveness as a teaching system has not been completed.

ABSTRACT: Planning and Training for Effective Use of Computers

Dr. Sandra Crowther, Microcomputer Coordinator, Michel Eltschinger, Linda Hyler, Classroom Teachers, Lawrence Public Schools, Lawrence, KS 66044

Microcomputer training has been a key ingredient in the planning and implementation of effective microcomputer use in the Lawrence, Kansas, school district.

During the 1980-81 school year the district had five computers which introduced several teachers and students to the power and possible uses of the computer. The district formed a microcomputer committee to make recommendations for purchases, training, and use. The committee recommended that the district hold the computers together for the first semester of 1981-82 so that an adequate number of staff could receive training. Eighty-one teachers voluntarily participated in four different training sessions. During the second semester the computers were placed in the various schools.

Demand for courses with credit and various interests have led the district to work in cooperation with the local university in making courses available and to also provide various opportunities for adding depth and breadth to staff's knowledge and skill through workshops. Separate workshops have been offered to meet needs of administrators and classified personnel.

This session will cover ideas for planning computer inservice, the training materials developed, and the successes and weaknesses encountered in the process.

AN OVERVIEW OF THE MATHEMATICS NEEDS OF COMPUTER SCIENTISTS

Anthony Ralston
SUNY at Buffalo
4226 Ridge Lea Road
Amherst, NY 14226
716-831-3065

The physical world is a mathematical object. This means only that physical scientists and engineers do--indeed, must--use mathematics to describe the phenomena with which they deal.

Unlike the physical sciences, computer science is concerned almost exclusively with non-natural (unnatural?) phenomena, namely, artifacts--algorithms and programs--which are human-created. Nevertheless it is now a commonplace that algorithms and programs are mathematical objects which can only be well-constructed, well-used and fully understood through the use of mathematics. Computer scientists may not agree whether the most effective approach to program verification is a purely formal one or one that involves informal reasoning (as, by the way, is the case with most proofs in mathematics). But they do agree almost universally that a competent practicing professional programmer or computer scientist must have a firm grasp of the mathematical tools needed to construct and analyze those mathematical objects called programs or algorithms. Why is it then that there appears to have been a trend in recent years away from mathematics in computer science and data processing programs? What, if anything, should be done about this?

It is wise to be wary of the opinions of research scientists in any discipline on what is most appropriate for undergraduate education in that discipline. Researchers too often view the chief function of the educational process to be reproduction of their own kind whereas what is most important for preparation for research may be quite different than what is essential for the always much larger group of students who will not go into research. In computer science, however, the subjects of research are still so close to the professional practice of the discipline that the curricular needs of research training are not materially different from those of all students in computer science. Unfortunately those most active in computer science curriculum matters in recent years have, with few exceptions, not been in close touch with research developments. The result has too often been curriculum recommendations which, as far as mathematics is concerned, have been out of touch with reality and, if followed, may well be harmful to the education of many, many students.

Do I imply that you must know much mathematics to be able to program? No, of course not. Or even that you need mathematics to write some correct programs? No, again. But I do claim that, to become a good programmer as well as a good computer scientist more generally, you must know a considerable amount of mathematics. For otherwise you will be unable to analyze your algorithms or write correct or efficient large programs or, in general, to be successful in writing large applications or systems programs. (See Gries [1981] to get the flavor of the kind of mathematics and mathematical reasoning which are needed for the effective development of computer programs.) Now it may be true that most of your students will spend their professional lives doing the equivalent--whatever that may be in 2025--of writing payroll programs in Cobol. But your obligation as a teacher is to give those of your students with the inherent capability to rise to higher level, more demanding tasks the basic educational tools to do so. If your graduates are mathematically illiterate or nearly so, many will be doomed to low-level positions who would otherwise have been able to rise higher.

What mathematics, then? While any mathematics in an undergraduate computer science curriculum is better than no mathematics, the traditional calculus sequence is just not relevant to virtually all undergraduate computer science courses. Conversely--and a fact still recognized by far too few--the very first course in computer science and its immediate successors would be much enhanced by a mathematics corequisite or prerequisite (one or two semester) course which gives the student a firm grounding in algorithmics generally, in the use of mathematical induction and in basic combinatorics and

discrete probability to name only a few of the possible subjects for such a course.

Even if you agree with my thesis, you may well say that, since computer science departments are so swamped with students and since mathematics departments don't offer the kinds of discrete mathematics courses needed by computer science students, requiring some calculus is the best you can do. Fair enough but be of good heart. The world of mathematics is changing (see, for example, Ralston and Young [1983]). I anticipate that, before many more years have passed, there will be good mathematics courses at the freshman level which cater, in part at least, to the needs of computer science students. Until that time comes, the least you can do is to make sure that the programs you offer make it clear that a healthy dollop of mathematics is a sine qua non for anyone who wants to reach his or her potential as a professional computer scientist.

REFERENCES

Gries, David [1981]: The Science of Programming, Springer-Verlag, New York.

Ralston, A. and Young, G.S. [1983]: The Future of College Mathematics: Proceedings of a Conference/Workshop on the First Two Years, Springer-Verlag, New York.

MATHEMATICS IN COMPUTER SCIENCE, AND THE APPLICATIONS PROGRAMMER

A.T. Berztiss

Department of Computer Science
University of Pittsburgh, Pittsburgh, PA 15260

Computer science tries to answer these questions:

What can be computed?
How fast can it be computed?
Is the computed result what we think it is?

Computability theory and complexity theory deal with the first two questions. In the third we ask whether a program is reliable. The results produced by computability theory are important because they provide a base on which all research in computer science ultimately rests, but it is unlikely that the existence of solutions will be a practical concern to the applications programmer. This leaves reliability and complexity. The purpose of this brief position statement is to point out the role of mathematics in the study of the two topics, and to indicate their relevance to the applications programmer, in particular the data processing specialist. Reliability, of course, is the more fundamental of the two--it should make no difference how rapidly we can compute an erroneous result. However, given that a program is reliable, we want it as fast as can be. We also want to minimize space requirements. This, too, is a concern of complexity theory, but space complexity is generally not as critical as time complexity.

The main purpose of complexity theory then is to provide an estimate of the time that an algorithm will take, expressed as a function of the size of the input. Since sorting is the most pervasive data processing activity [1], the most relevant result provided by complexity theory for data processing is that sorting of n items is at least an $O(n\log n)$ process. But what does $O(n\log n)$ mean? This is a measure of the time required to sort the n items. We call it the time complexity of sorting, read it as "of the order of $n\log n$," and interpret it as saying that the time required to sort n items increases as $n\log n$ with an increase in n. This is an oversimplified interpretation, but a more complete explanation requires knowledge of functions as mathematical objects, of asymptotic approximation, and, in this instance, of the nature of logarithms. Unfortunately, without a proper understanding of the meaning of the O-notation, there is the danger of misinterpretation. For example, there exist matrix multiplication algorithms that are $O(n^k)$, where k is below the conventional 3, but n would have to be very large indeed before their use would be justified. Returning to sorting, the time complexity of the well known quicksort algorithm is $O(n\log n)$ on the average, but $O(n^2)$ in the worst case. Rudimentary knowledge of probabilities and their distributions would help one appreciate these results.

In discussions of the complexity of algorithms one often hears the term *combinatorial explosion*. What it means is that when we generate a set of derived objects from a set of n basic objects, the growth of the number of derived objects is prohibitively fast as n increases. For example, it it is not easy to believe that the number of ways of arranging 10 books on a shelf is as high as 3,628,800. Proper appreciation of combinatorial explosion can be gained only by the study of some combinatorics, by working with the large numbers.

Problems are sometimes separated into those that can be done in polynomial time, and those that require exponential time. The complexity of a polynomial-time algorithm is $O(n^k)$, where k is some positive integer; the complexity of an exponential algorithm is $O(k^n)$, or even $O(n^n)$. For example, the number of ways of arranging n books on a shelf can be expressed approximately as $O(n^n)$, which means that an algorithm for generating all these configurations would have to have at least this time

complexity. We also speak of *hard* problems. Experience shows that such problems sometimes require prohibitive amounts of time for their solution. Many of the hard problems belong to a class we call the class of NP-complete problems. The distinguishing feature of this class is that if any one NP-complete problem could be solved in polynomial time in the worst case, then all NP-complete problems could be solved in polynomial time. We suspect that NP-complete problems require exponential time in the worst case, but, even if this were not so, existence of a polynomial time algorithm for a problem does not necessarily mean that solving the problem is always practicable. The following table bears this out. It expresses n^3, n^6, and 2^n microseconds in more convenient time units for various values of n. We soon reach a value of n at which an $O(n^6)$ algorithm is impracticable.

n	n^3	n^6	2^n
20	0.016 sec	4.1 min	33.5 sec
50	0.125 sec	4.3 hrs	35.6 yrs
100	1.000 sec	11.6 days	4×10^{16} yrs

NP-completeness pervades all of programming, including data processing. Out of the vast set of NP-complete problems the following four have definite bearing on data processing. Decision table optimization is the first. If it is possible to associate probabilities with the outcomes of evaluations of decision table conditions, then the evaluation of the conditions can be arranged in a sequence that minimizes some cost function. Unfortunately the finding of the optimal sequence is NP-complete [2]. So is the similar problem of organizing a trie dictionary to minimize storage requirements [3]. The fact that the selection of an aesthetically satisfying layout of a binary tree is a hard problem [4] suggests that layout problems in general will be difficult. Finally, the problem of deciding whether a relation schema of a relational data base is in Boyce-Codd normal form is NP-complete [5].

No matter what importance we give to complexity, reliability is more important. The writing of unreliable programs should be regarded as professional incompetence, but we still find it very difficult to produce reliable programs despite a variety of approaches that have been tried to improve this state of affairs. The first attempts were to codify sound programming practices as sets of maxims (see, e.g., [6]). The maxims were elaborated into design techniques that have become known as software engineering (see, e.g., [7]). The culmination has been a rigorous formal program development process [8,9]. In the latter case we speak of *program verification* or *program proof*, and all programmers should become familiar with this approach.

What is a program proof? In mathematics one proves a theorem by showing it consistent with a set of assumptions (axioms). Expressed in other words, we transform the assumptions into a theorem. In doing this we make use of well defined laws of logic, the rules of inference. The axioms and the rules of inference define the mathematical system in which we work, and every statement that is derivable from the axioms by the rules of inference is a theorem. In practice, however, we find most such statements uninteresting; generally our approach is to set up an interesting statement that we believe to be true as a hypothesis, and to show that the hypothesis is indeed true, i.e., that it is a theorem. In program proving, we start with some description of the properties of the input, and from this try to derive a statement describing the required output. Expressed in very simplistic terms, we regard the description of the input as an axiom or a set of axioms, the statement types of a programming language as rules of inference, and the description of the output as a hypothesis. As we move down a program, the statements in the program are interpreted as applications of the rules of inference that are to transform the description of the input into a description of the output. The program transforms the input into an output; the proof transforms a description of the input into a description of the output. If the description of the output is what we want it to be, then we conclude that the program is in fact correct.

How does this affect the programmer? For one thing, structured programming arose out of program correctness concerns, and, even if we do not get far with formal program proofs, structured programming alone has been a worthwhile achievement. It has certainly made programs easier to understand and maintain. Second, programming languages are deliberately being made simple because this facilitates program proofs. Pascal and--despite appearances--Ada are examples of this trend.

The input and output descriptions constitute a specification of the program. They are usually expressed as assertions in the first order predicate calculus. It is unlikely that many programmers will be called upon to prove programs correct in the formal sense, but some time in the future most programmers will be seeing program modules that have been certified to be correct, and it will be necessary to be able to read and understand the specifications. Moreover, even an informal proof of one's program can bolster one's confidence in the program.

However, the general outlook for program proving is far from rosy, and the technical background of an applications programmer should be adequate to understand why some of the claims that are being made for program proving techniques are unrealistic. The difficulties arise from complexity analysis. To give just one example, the length of an automatic formal proof of a program is exponential in the size of the program [10].

The topics mentioned in this shallow survey are treated at length in recent books. Program proving is surveyed by Berg *et al.* [11]. Horowitz and Sahni [12] provide a good introduction to the complexity of algorithms; NP-completeness specifically is surveyed by Garey and Johnson [13]. A study of these mathematics based theoretical topics will provide data processing specialists with better insight into the nature of data processing. Moreover, it will enable them to exercise critical judgement in dealing with the more exaggerated claims for some theoretical techniques. A more extensive discussion of the relevance of computer science theory to data processing specialists can be found in [14].

REFERENCES

1. DANIELS, A. and YEATES, D., eds. (1969): "Basic Training in Systems Analysis", Pitman, London.

2. HYAFIL, L. and RIVEST, R.L. (1976): "Constructing optimal binary decision trees is NP-complete", Inf. Proc. Letters, vol.5, pp.15-17.

3. COMER, D., and SETHI, R. (1977): "The complexity of trie index construction", J ACM, vol.24, pp.428-440.

4. SUPOWIT, K.J. and REINGOLD, E.M. (1983): "The complexity of drawing trees nicely", Acta Informatica, vol.18, pp.377-392.

5. BEERI, C. and BERNSTEIN, P.A. (1979): "Computational problems related to the design of normal form relational schemas", ACM Trans. Database Syst., vol.4, pp.30-59.

6. KERNIGHAN, B.W. and PLAUGER, P.J. (1974): "The Elements of Programming Style", McGraw-Hill, New York, NY.

7. ZELKOWITZ, M.V., SHAW, A.C., and GANNON, J.D. (1979): "Principles of Software Engineering and Design", Prentice-Hall, Englewood Cliffs, NJ.

8. JONES, C.B. (1980): "Software Development: A Rigorous Approach", Prentice-Hall, Englewood Cliffs, NJ.

9. GRIES, D. (1981): "The Science of Programming", Springer-Verlag, New York.

10. JONES, N.D. and MUCHNICK, S.S. (1981): "Complexity of flow analysis, inductive assertion synthesis, and a language due to Dijkstra", in Program Flow Analysis: Theory and Applications (Muchnick and Jones, eds.), Prentice-Hall, Englewood Cliffs, NJ, pp.380-393.

11. BERG, H.K., BOEBERT, W.E., FRANTA, W.R., and MOHER, T.G. (1982): "Formal Methods of Program Verification and Specification", Prentice-Hall, Englewood Cliffs, NJ.

12. HOROWITZ, E. and SAHNI, S. (1978): "Fundamentals of Computer Algorithms", Computer Science Press, Potomac, MD.

13. GAREY, M. and JOHNSON, D. (1979): "Computers and Intractability: A Guide to the Theory of NP-Completeness", Freeman, San Francisco, CA.

14. BERZTISS, A.T. (1983): "Data processing and computer science theory", Proc. ACM SIGCSE 14th Tech. Symp. Comp. Sc. Education (ACM SIGCSE Bulletin, vol.15, no.1), pp.72-76.

MATHEMATICS SERVICE COURSES FOR THE COMPUTER SCIENCE STUDENT

Martha J. Siegel

Department of Mathematics and Computer Science
Towson State University
Towson, MD 21204

The Mathematical Association of America (MAA) has established a Panel on Service Courses whose members are drawn from the Committee on the Undergraduate Program in Mathematics (CUPM) and the Committee on the Teaching of Undergraduate Mathematics (CTUM). It has been my responsibility, as a member of the Service Course Panel, to investigate the mathematical needs of computer science majors. One can turn to the organizations which have assumed the responsibility for recommending curriculum for such students. I will concentrate on ACM's Curriculum '78. With regard to the mathmatics requirement, the ACM Committee on the Curriculum for Computer Science stated in its report:

> An understanding of and the capability to use a number of mathematical concepts and techniques are vitally important to a computer scientist. Analytical and algebraic techniques, logic, finite mathematics, aspects of linear algebra, combinatorics, graph theory, optimization methods, probability, and statistics are, in various ways, intimately associated with the development of computer science concepts and techniques. ... Unfortunately, the kind and amount of material needed from these areas for computer science usually can only be obtained, if at all, from the regular courses offered by departments of mathematics for their own majors.

Mathematicians are sensitive to these comments. Recently, computer scientists and mathematicians have been cooperating in developing suitable courses for the first two years. There is considerable agreement that discrete mathematics can be introduced at the freshman level. Several pilot projects, funded by the Sloan Foundation, will develop courses with material of this type that is either integrated with material from the standard calculus course or is a stand-alone companion to calculus. Clearly, the topics eventually included in the courses will significantly affect upper-division mathematics courses.

However, something has already happened to change mathematics at the upper division. In 1981, CUPM issued its Recommendations for a General Mathematics Sciences Program. Course requirements in 1978 could hardly have reflected this. Curriculum '78, relying on previously recommended CUPM courses, listed the following as required for all computer science majors:

MA1	Introductory calculus
MA2	Mathematical analysis I
MA2P	Probability
MA3	Linear algebra
MA4	Discrete structures

and for some students, also

MA5	Mathematical analysis II (multivariate calculus)
MA6	Probability and statistics

Numerical analysis was listed as a computer science course and not required in the core curriculum.

The Curriculum Committee, chaired then by Dick Austing, gives the distinct impression that these courses are not really what was wanted, but that the committee was not about to redesign the existing mathematics curriculum. Now that CUPM recommendations have led mathematicians to modernize their curricula, are the ACM goals being met more satisfactorily?

Let us assume that a one year course in discrete mathematics is in place for freshmen. Assume that computer science majors take at least one year of calculus. The calculus recommended by CUPM is quite different in flavor from what had been traditional. The course, designed by CUPM for mathematics majors as well as others, is standard in the topics selected for the first semester, but they are to be taught with heavy emphasis on models and applications. In the second semester, however, twelve of the forty lecture hours are to be used to achieve a computer emphasis. This will mean that there will be an early introduction of numerical methods. The techniques of integration are de-emphasized, while applications of the integral using a modeling approach are introduced. The treatment of sequences and series should also change. Sequences, the report suggests, should be defined not only through "closed" formulas, but also via recursion and iterative algorithms. Rates of convergence and error analysis should be highlighted. Power series should be stressed and the use of Taylor series as approximations with the accompanying computer implementation introduced.

What follows? I have taught statistics for twenty years, and am convinced that students cannot learn enough about probability or statistics in less than a semester. Even if some topics are included in the freshman course, computer scientists who want to deal effectively with measurement and

and evaluation of programs and systems, operating systems theory, or canned statistical packages, need more. New CUPM recommendations include a nearly perfect course. It is a post-calculus course in statistical methods. The emphasis is on data collection, data organization and description, probability, statistical inference, computer simulation, and an introduction to statistical packages. We offer this course at Towson State, and find it is not only a service course for computer science majors, but to those in other disciplines as well as the mathematics major.

Linear algebra courses have been evolving for some time. The CUPM recommendations address many of the concerns that it may have failed as a service course. For example, the suggestion is that the course make heavy use of models and applications. Theory has not been abandoned, however, and my own observation has been that this course is perfect for an emphasis on the algorithmic approach. In addition, the course should introduce computational methods as well as abstraction and accessible proofs.

The recommendation of the ACM for a discrete structures course cannot be met entirely by a course at the freshman level. Although we do not yet have results of pilot projects, I suspect that many of the course outlines are a bit ambitious for the ordinary freshman whose high school preparation in the area of abstraction is practically nil. I believe that a junior level discrete mathematics course may still be an attractive and necessary course for both computer science and mathematics students.

One omission from the ACM list of courses is one in operations research and/or mathematical modeling. The objectives of such courses are listed as desirable in ACM's report and are not met by other previously mentioned courses. The CUPM recommendations include several courses of this type, and my experience in teaching one for about six years is that this is a superb way to teach students how to abstract salient features from practical and complicated problems, and to show them how to apply many mathematical and computational skills to a single problem. Here is a place in the curriculum for the enhancement of their communications skills; where they can write a meaningful paper and give a presentation on a large project. Their ability to use linear algebra, probability, statistics, differential equations, numerical methods, graph theory, linear programming and graphics can make students feel that they have a command (albeit elementary) of powerful, beautiful and complementary subjects.

Mathematics is evolving, our course offerings are changing, and rather than growing further apart, mathematics and computer science can be seen as helpmates.

REFERENCES

(1) "Curriculum '78, Recommendations for the Undergraduate Program in Computer Science", Communications of the ACM, v. 22, n. 3, March, 1979, pp 147-165.

(2) Recommendations for a General Mathematical Sciences Program, Committee on the Undergraduate Program in Mathematics, Mathematical Association of America, 1981.

STIRRINGS IN THE MATHEMATICS CURRICULUM:
CHANGES MATHEMATICIANS ARE THINKING OF MAKING

Stephen B. Maurer

Mathematics Department, Swarthmore College, and
The Alfred P. Sloan Foundation

Other members of this panel are speaking to the question of what mathematics a computer scientist needs to know. It is not exactly the same mathematics as mathematics departments teach. This suggests that mathematicians ought to think about changing what they teach. I am here to report that some of them are thinking about this, and to indicate what they are thinking.

I should begin by acknowledging that mathematicians have a love-hate relationship with the other disciplines which they service. On the one hand, mathematicians are very pleased that so many other fields recognize that they need mathematics, and pleased that students from those fields who want mathematics training provide a steady source of employment in teaching for mathematics faculty and graduate students. On the other hand, mathematicians are suspicious of the motivation of these serviced students, believing they are merely vocationally oriented, only want to know how to use techniques, and have little interest in really understanding or appreciating mathematics. Consequently, when another department asks that certain mathematical topics be taught for the good of their students, the typical math department reaction is either to:

1) Do nothing, in which case the other department begins teaching the subject itself; or

2) Make up a special course for students from that department, thus not letting the new material have any effect on the mainstream mathematics curriculum.

We can all sight examples where one of these has happened. And when the other department is one whose enrollment is growing rapidly, and threatens to siphon off most of the students who would otherwise major in mathematics, well, the love-hate feelings are that much stronger!!

The way to get a major change into the mainstream mathematics curriculum is to convince mathematicians that the change is of value to students majoring in a variety of disciplines, including mathematics itself. If there is a new subject, or a new point of view, which mathematicians come to believe is central for understanding both mathematics and applications, then things may move.

Today we are in the unusual position that many mathematicians have begun to think that something this important has come along - the algorithmic point of view. I probably don't have to explain what I mean by "algorithmic point of view" at a computer conference. Let's just say that when a mathematician thinks about some mathematical object or operation, he (or she) has an algorithmic view if he doesn't merely ask if the thing exists, but asks how to find it, how to find it systematically, and how many steps his procedure takes. Such a viewpoint applies to very elementary problems, like multiplying two numbers, as well as to high-powered abstract mathematics. It adds new life, and uncovers new unsolved problems, even in elementary mathematics. For instance, nobody knows the most efficient way to multiply matrices. It is known that the standard method -- row i of one matrix versus column j of the other - is not the best. There are other methods which take fewer real-number multiplications. (However, these methods are much more complicated to understand and program, and thus they are of little use, at least so far, for the small matrices usually multiplied in practice.)

This algorithmic point of view is not actually all that new. When ancient man, confronted with the problem of representing, adding and multiplying numbers, developed the abacus, Roman numerals and the decimal system, he was dealing with algorithmic questions. Mathematical induction, the classical logical method for proving all cases by proving how to get from one case to the next, is intimately related to iteration and recursion, perhaps the key ideas in modern algorithmic

thinking. Recursive definitions are also old hat in mathematics. However, modern developments make these topics much more important, and tie them in with new topics. With modern computer languages, one has to have a much more precise concept of an algorithm than one used to. With the speed of modern computers, many problems which were unthinkable to attack computationally before are now tractable. On the other hand, many still are not. So one needs to concentrate much more mathematical energy on understanding when an algorithm works and what is its order of complexity (i.e., number of steps it takes).

I have just argued that the algorithmic point of view expands an old view in mathematics and reinvigorates many areas of mathematical investigation. For this reason it certainly belongs in the math curriculum. One can go further, though. It has been claimed, rightly I think, that such habits of thought as recursive thinking (which includes reducing to the previous case and seeing dynamically how the current stage arises from the previous) and structured thinking (top-down thinking, step-wise refinement, etc.) are major components of successful human thinking in almost any problem-solving endeavor. Therefore, it is useful for everybody to see embodiments of these habits of thinking, in math courses and computer courses.

The sort of mathematical material I am talking about is usually referred to (by others and by me) as discrete or finite mathematics. However, this phrasing can be misleading. Finite mathematics courses were introduced 25 years ago, and at that time they had neither computer science students nor computing as a motivation. In fact, they were usually aimed at social science students. While it is true that most of the topic names in the new discrete mathematics courses being proposed are the same as in the old course, the point of view and the examples today should be rather different. For example, one of the topic names is "counting methods". In the old course, counting methods would include the concept of a combination of n things taken k at a time. One finds a formula for the number of such combinations and applies this formula in many counting problems. In the new course, one would also ask: How can one generate a random combination, or a sequence of random combinations, given that a computer has a random number generator but not a random combination generator? Also, in the new course one would pay much more attention to difference equations (equations relating one term in a sequence of numbers to several preceding terms, a type of equation often not covered at all in the old course), because if one wants to count the number of steps performed by a computer program with a loop or a recursive call, one immediately gets a difference equation to solve. In the new course, several examples of such computer-related applications would be given.

Another topic name common to both courses is "graph theory". In the old course, one might prove Euler's theorem that, in a connected graph (network) with an even number of edges at each vertex, one can trace a path over all the edges without lifting one's pencil or repeating any edges. In the new course one would pay careful attention to how one might explicitly find such a path by an algorithm, and to how one can prove the theorem by first stating the algorithm and then analyzing how it terminates. In the old course, one typically proved this theorem by a more "existential" method, say, proof by contradiction: one supposes there is a counterexample graph, existentially imagines picking the counterexample with the smallest number of edges, and then finds a contradiction by considering a related graph with fewer edges which by hypothesis does have the right sort of path.

In short, one cannot assume, just because a math department offers a finite or discrete math course, that the department is meeting the challenge of incorporating the new algorithmic needs into its curriculum.

I have explained why there ought to be stirrings in the mathematics community. What stirrings are there?

Most universities and many colleges now have an upper level course in discrete structures for computer science students, given either by the CS department or the math department, depending on which of 1) or 2) above prevailed. The stirrings I am talking about are additional stirrings towards offering a broadly targeted course, intended for freshmen and sophomores.

There are already a smattering of courses like this around the country. When there are a lot of good reasons for trying something new, some faculty at some schools just start doing it, without any prompting or funding from outside. When the Mathematical Association of America (MAA) asked in its newsletter last September to hear from faculty who were giving new courses of this sort, several people wrote in.

There has also been an increasing

amount of writing on the subject. One of the most forceful writers has been Tony Ralston of this panel. One of his first articles on the subject was titled "The Twilight of the Calculus". Although it seems he has now recanted (in part), he certainly got people thinking.

Another sort of writing which is very important is text writing. Unfortunately, texts for the new sort of course described above are not yet available in print. Fortunately, several are in preparation.

I have 3 more activities to report. First, a whole conference has now been held on this subject - last June at Williams College, organized by Tony and funded by the Sloan Foundation. The proceedings are out, titled "The Future of College Mathematics", published by Springer-Verlag. The articles are excellent and wide-ranging. Many of the articles prepared in advance discuss the potential value of discrete mathematics to students who major in discipline X, where X takes on such values as computer science, mathematics, physics, engineering, management science and social science. Also included in the proceedings are the reports of several workshops held at the conference. One workshop outlined a mathematics curriculum for the first two collegiate years consisting of a 1-year continuous math course and a 1-year discrete math course. Another workshop outlined an integrated 2-year program, consisting of various 5-week modules which could be selected and ordered in various ways for various students.

I urge you all to read these Proceedings. I believe they will have a wide impact.

Second, the Sloan Foundation has been sufficiently persuaded by the stirrings so far that it has decided to run a program of grants to mathematics departments willing to try major reorganizations of their first two years of courses. This paper is being written shortly after letters went out to approximately 30 colleges and universities, inviting them to apply for one of 5 grants of up to $40,000. Announcements of awards should be made by June 15.

Finally, the Committee on the Undergraduate Program (CUPM) of the MAA has set up a panel to study the development of a new curriculum for the first two years. This CUPM panel will both follow experiments going on and make proposals about what more to do. (See the paper by Martha Siegel, one of my co-panelists at this NECC conference. She is the chairman of the CUPM panel I refer to.) The CUPM panel includes members from ACM and from the American Society for Engineering Education. The MAA is the foremost American organization concerned with collegiate mathematics, and the work of CUPM is one of its most respected activities. Reports of CUPM panels have had significant impact on curricula before. We have good reason to expect this to happen again.

By the time this paper is presented, perhaps the stirrings will be rumblings. I look forward to giving an update in person.

USING A LARGE SCREEN COMPUTER SYSTEM TO IMPROVE TEACHING

David R. Lundstrom
Mahlquist Jr. High School
Plain City, Utah 84404

ABSTRACT

The purpose of a large screen computer system is to increase the quality of instruction in the classroom and to reduce the teacher's workload. The heart of this system is a computer controlled, ten foot television screen which functions as a fully automated, electronic blackboard and display for an entire class. It also produces the majority of the science curriculum materials and controls the management of most classroom records.

The system functions as follows: all large group instruction materials such as charts, diagrams, illustrations, and lecture notes are instantly displayed with the touch of a button. This is accomplisned through detailed graphic pictures and text which appear in color and animation to a whole class at a time. More traditional curriculum media such as transparencies, opaque projections, and chalkboard drawings are now created and stored electronically with a computer. These materials are recalled and revised, displayed or printed at any time. This includes all printed materials such as worksheets, handouts and tests. The system also manages and stores classroom records on computer.

The overall effect of a large screen computer system is to place the teacher in an almost totally automated, electronic classroom that provides higher quality science instruction to the student while actually reducing the teacher's workload. The teacher, however, remains as the essential teaching element in the classroom. In addition, student motivation and enthusiasm are increased and discipline problems are reduced. The classroom becomes a significantly better place to learn and to teach for the students and for the teacher.

Educational Software Copyright Issues

Ronald E. Anderson, Chair
University of Minnesota

ABSTRACT

This session follows up issues raised at the highly popular NECC '82 session entitled "Impact of Copyright Laws on Computers in Education." The panel will address the implications of new technology such as classroom networks for copyright legislation. A special emphasis will be placed upon the perspectives of the educational publishing industry and what alternative solutions such as site licensing agreements and backup copies are available. The session will begin with a review of current software copyright developments, then the panel will be asked to recommend how educational institutions can cope with restrictions on software copying.

PANELISTS

Carol Risher
American Association of Publishers

Kenneth E Brunbaugh
Minnesota Educational Computing Consortium

David Edwards
McGraw-Hill Publishing

Scott Mace
Infoworld

An Anonymous Software Copier

SPONSORS

SIGCAS
SIGCUE

Teaching Structured Programming in the Secondary School

Jean B. Rogers
Computer and Information Science
University of Oregon
Eugene, OR

SPONSOR: ICCE

ABSTRACT

This session will consist of various approaches to teaching programming in secondary schools. Samuel F. Tumolo will discuss teaching structured programming using Pascal. Pascal was designed primarily to aid in the teaching of good programming style. By its' design, it makes learning structured programmin easier that many other computer languages do.

Procedures and other constructs of Pascal make it easier to develop good programming habits and style. The features of Pascal that aid in the development of structured programming atmosphere will be discussed.

There are characteristics of the Pascal environment that may detract from its effectiveness. Their effect on structured programming will be explored.

Finally, problems that secondary school students encounter in learning Pascal will be discussed.

Structured language alternatives will be discussed by Michael Ward. With the hardware and software developments of the last ten years, the ways that programming is taught and learned have changed. For the most part, serial batch environments are no longer the norm. So once initial familiarization with the system is gained, the methods by which programs are created, tested and corrected are much less cumbersome and awkward. Therefore, there is one less impediment to learning.

A much more important development has been the discipline of structured programming design as a problem solving device and structured languages as a means of implementing those solutions. As the languages move from the artificial statements and structures of the machine to the more natural human like native language another major impediment has been removed.

Students learn more, quicker. They are actually able to solve much more complex problems sooner than when they were using the old methods, if they were able to solve the problem using the methods at all. Pascal is one of the above mentioned languages and is not without fault It solves many of the old problems but in the process creates some new ones. There are some BASIC and FORTRAN based alternatives that facilitate the development of good problem solving habits as well as force the student into the discipline of structured program design.

Language independent instruction will be discussed by Anthony Jongejan. Pascal is not available to students in most high schools today for many reasons, including extra cost, hardware and the inadequate preparation of teachers.

Currently, the reality in most high school computer science programs is that BASIC will be taught. Thus, every attempt must be made to convince teachers of computer science to incorporate the problem solving model and as many structured programming concepts as possible in their teaching of programming when using BASIC.

Concepts that should be incorporated when teaching programming using BASIC include meaningful variable names, indenting your programming listing to show the scope of various program constructs, the orderly use of the GOTO statement and documentation. The modular solution of the problem utilizing GOSUB's when programming that solution should be emphasized.

Finally, it may be desirable to implement the "while - do" loop, "repeat - until" loop, "for - next" loop and the "if - then - else" statement ir BASIC and emphasize their use.

With full knowledge that this will not replace Pascal for implementation in an appropriate structured language, this is a partial solution for those teachers who do not have the resources to offer Pascal

PARTICIPANTS:

Samuel F. Tumolo
Cincinnati Country Day School
Cincinnati, OH 45243

Michael Ward
Willamette University
Salem, OR

Anthony Jongejan
Everett High School
Everett, Washington

Nationwide Computer Literacy Project

Daniel Updegrove
EDUCOM
Princeton, NJ 08540

Steven Gilbert
EDUCOM
Princeton, NJ 08540

ABSTRACT

In response to the expressed needs of many of its member colleges and universities, EDUCOM has begun a major project on computer literacy in higher education. Through mail surveys, site visits, literature searches, conferences, electronic mail and computer conferencing, EDUCOM seeks to determine the current state of the art in computer literacy programs for students, faculty, and staff. EDUCOM plans to publish survey results; provide evaluation criteria, exemplary models, and guidelines; and develop consulting teams to assist colleges and universities in creating or upgrading computer literacy activities.

The NECC presentation will (a) provide a progress report on the project, including the initial survey results; and (b) describe how attendees can participate in the project through task forces, electronic communication, and computer based project activities.

Several recent studies document a real and urgent national need for college graduates who have had a solid introduction to information technology, but three primary obstacles exist:

1. No coherent definition and conceptual framework for computer literacy have gained wide enough support to provide criteria for evaluating present instructional programs and to direct the development of new ones.

2. Higher education has no effective feedback system for identifying successful computer literacy program models in some institutions and for adapting and disseminating them to others.

3. A flood of vendor claims about new hardware, software, and "courseware" overwhelms the limited capacity of most individuals, and even institutions, to monitor the progress of information technology on their own.

This is a project of great scope and potential impact, focused on a complex need. Consequently, we have designed the project with four categories of participation:

1. Interested observers will receive periodic announcements of project activities, direction, and interim results. They may occasionally submit reactions, suggestions, interesting articles, reports of noteworthy local programs, samples of materials, etc., (to project staff via phone or mail).

2. On-line participants will participate in the same way as interested observers, but will also join in one or more forms of electronic project activities; e.g., exchanging news and ideas via electronic mail, helping with the ongoing development of computer literacy data bases (literature, people, programs, etc.) and/or contributing to a computer conference.

3. Implementation Task Force members will represent their institutions (both when attending meetings and when on home campuses) by being available for consultation and assistance with key project activities; e.g., developing and testing survey instruments, organizing site visits to noteworthy computer literacy programs, commenting on the effectiveness of materials or curricula on nearby campuses, etc.

4. Leadership Task Force members will join other nationally recognized leaders (from EDUCOM member institutions, form information industries, from other businesses, from government agencies, etc.)

who are ready and able to commit insight, time, and authority to developing a compelling new conceptual framework, set of definitions, and program evaluation criteria for computer literacy in higher education. The will also provide overall policy guidance and advice to the EDUCOM staff and the Implementation Task Force.

SPONSOR: EDUCOM

Using the Microcomputer Creatively with Young Children

Marilyn Church
June Wright
University of Maryland
College Park, MD 20742

ABSTRACT

This tutorial will address the question, "Do Microcomputers Enrich the Preschool Environment?" It will report the findings of a two year pilot project which introduced three, four, and five year-olds to microcomputers. The presentation will focus specifically on the techniques and programs developed as a result of that project. A preschool curriculum based on the Logo philosophy which gives the child a sense of mastery will be demonstrated. The relationship of this curriculum to the introduction of Logo Language in kindergarten and primary grades will be explained.

A videotape showing children interacting with the microcomputer will highlight the discussion of the kinds of early learning which the use of the microcomputer offer. Recommendations for fostering creativity through the discovery approach will be included. A consideration of the role of the parent in computer education and the concept of the teacher, the child and the programmer functioning as a team will complete the presentation.

HUNTINGTON III: MICROCOMPUTER COURSEWARE DEVELOPMENT PROJECT

by Thomas T. Liao

Department of Technology and Society, State University of New York
at Stony Brook, Stony Brook, New York

Abstract

In this paper, an overview of the Huntington III Project is presented. The primary objective of this National Science Foundation-funded project is the development of a set of interactive courseware modules for use in grades 8-12 mathematics and science classes. The project is an outgrowth of the Huntington I and Huntington II computer simulation projects, which were also funded by NSF.

Introduction

Huntington III is developing courseware for the Commodore PET and Apple II microcomputers because of their widespread use in education and their graphical characteristics. The interactive science and math courseware packages are of two types:

Multiple Use Courseware

These courseware packages feature generic programs that can be adapted for satisfying various instructional needs; for example, an educational game such as Tic-Tac-Flex helps students learn simple arithmetic skills, algebraic equations, or elements on the periodic table. Also teachers can decide the version of the game and questions that they want their students to use.

Applications Courseware

These courseware packages focus on technological topics that will help to provide opportunities for students to apply and integrate basic science, math, and programming concepts; for example, an electrical energy inventory program helps students to learn some basic physics concepts and algebraic equations that are used in analyzing a real-world program.

A comprehensive teacher's guide accompanies each computer program, in which the rationale and performance objectives are clearly stated and the operational pattern and sample runs are provided. Also included are master copies of student handouts and worksheets and recommendations for instructional strategies and additional resources. Each computer program is fully documented for its structure (flowchart and pseudocode), meaning of variables, and equations used. A complete program listing and suggested methods for adapting programs completes the teacher's guide. In addition to the exemplary courseware packages, the project staff plans to develop a courseware design handbook.

Dr. Thomas T. Liao, Department of Technology and Society, State University of New York at Stony Brook, is Director of the Project; Dr. Ludwig Braun, New York Institute of Technology at Old Westbury, is the principal collaborator. Initially copies of the exemplars will be available for those interested in field testing the materials.

Approach to Design and Development

The Huntington III Project uses a team-oriented systems approach and a well-defined set of design criteria. To help ensure quality courseware, the project uses a systems approach to instructional design and development and carries out the work with a team of two or three professionals who have the combination of expertise that is required.

The development of quality microcomputer courseware should be similar to the development of other types of instructional materials. An effective approach is one that is modelled after engineering systems design and development. The components of this systems approach to instructional design and development are:

1. Identification of instructional need(s) and characteristics of students.
2. Specification of design criteria and learning objectives.
3. Identification of constraints such as limitations of the learning environment and the microcomputer to be used.
4. Brainstorming of alternative designs of computer program(s) and support materials.
5. Specification of the content of a courseware package.

6. Development of a prototype courseware package.

7. Field testing of the courseware package.

8. Revision of the courseware package based on feedback from students and teachers.

The development of quality microcomputer courseware requires the creative combination of six areas of expertise:

1. Instructional design and development experience.

2. Knowledge of subject matter.

3. Understanding of pedagogical methods.

4. Knowledge of microcomputer capabilities.

5. Knowledge of programming.

6. Ability to write curriculum materials.

Since it is unlikely that any one person can be expert in all of the above areas, a team of two or three people works together to develop quality courseware. Our experience also demonstrates that the give-and-take that occurs among team members leads to the development of more effective materials.

In the Huntington III Project, we are guided by the preceding suggestions and, in addition, use a set of more specific design criteria that guide the design and development of each courseware package. These criteria are especially useful for fine-tuning the materials; some are related to the teacher's guide, while others deal with the computer program.

When developing a computer program, the team uses four sets of design criteria, which are related to the following four questions:

1. Is the program "user friendly"? To satisfy this criterion, program users should have complete control of the pace of the presentation. Not only should instructions and other screen output be easy-to-understand, but the programs should be designed so that they can be easily modified.

2. Is the program "user proof"? To satisfy this criterion, the program should be able to continue even if the user strikes the wrong key; the program should only accept meaningful input data.

3. Does the program take advantage of the unique characteristics of microcomputers? Appropriate use of graphics is one way of satisfying this criterion. Other techniques include intelligent use of the timing capability, provision of instant feedback, and use of simulation gaming activities.

4. Is the program highly interactive? The user should be consistently involved in providing responses to problem-solving activities. Learning should be as active as is possible.

A teacher's guide accompanies each computer program and contains information that: allows for effective use in the classroom or learning laboratory; is required for program modification; provides additional documentation so that the program can be used as an example of good programming.

An example of the table of contents of a teacher's guide follows:

1. Overview
   - This includes an abstract, specification of grade levels of users, microcomputers to be used, and contents.

2. Performance Objectives
   - These state what students should be able to do after using the courseware.

3. Rationale
   - This specifies why teachers should use this package.

4. Operation of Computer Program and Sample Runs

5. Student Materials
   a) Description
      - Description of handouts and sample responses.
   b) Handouts/Worksheets
      - Provision of background or guidance for the laboratory.

6. Instructional Strategies
   - Strategies include suggested ways for using the courseware package.

7. Documentation of Computer Program
   a) Structure (Flowchart and Pseudocode)
   b) List of Variables
   c) Mathematical Model(s)
   d) Listing of Program

8. How to Modify Program

9. Additional Resources
   a) Background Information
   b) Extension Ideas
   c) References

Field Testing of Prototype Courseware Packages

During the 1982-83 school year four prototype packages are being field tested--each computer program and the accompanying teacher's guide will be used by at least 10 teachers and in 20 classrooms. Abstracts of the four packages and the evaluation questions follow.

Besides obtaining feedback via the student and teacher questionnaire, the project staff will observe students and teachers actually using the materials. During the summer of 1983, the courseware developers will use the information obtained from field testing to revise existing materials and to develop additional exemplary courseware.

Abstracts of Some Huntington III Microcomputer Courseware Packages

Domestic Electrical Energy Use and Cost (Applications Courseware)

In this applications package, students interact with two computer programs to learn how much electrical energy is used in their homes and how their electric bill is computed. They also have the opportunity to explore the effect of various methods of conserving energy to minimize the size of their electric bill; in addition, they carry out data collection and analysis.

Yellow Light Problem (Applications Courseware)

This courseware package provides an opportunity for students to study the factors that affect the motion of a car at a traffic intersection. First, students can use a simulation to determine their response time, which is then combined with four other factors (speed limit, yellow light time, deceleration, and width of the intersection) to determine the Go, Stop, and Dilemma zones. By changing each of the five factors, one at a time, students can study the effect of each parameter. Finally, they can graphicially analyze:

a) How the speed limit, yellow light time, and width of the intersection affect the Go zone.

b) How the speed limit, response time, and deceleration affect the Stop zone.

Mass Spring (Multiple-Use)

When using Mass Spring, students interact with a computer program to learn how two user-controlled variables--spring constant (K) and damping (B)--affect the displacement of an object (M) over time (T). The resulting displacement of the mass is shown in graph format. The students also have the opportunity to decide what values of K and B are appropriate for specific mass-spring damping systems such as the suspension system of an automobile.

Tic-Tac-Flex (Multiple-Use)

This courseware package includes two drill-and-practice programs that are modelled after the tic-tac-toe game. Teachers can modify the programs in three ways: (1) choice of game format; (2) questions and answers to be used; (3) instructions to be used. The sample programs contain a review of chemical symbols and practice with algebraic relationships.

Concluding Comments

The objective of this paper, written in early December 1982, is to provide an overview and update of the approach and products of a microcomputer courseware development project. In June 1983, the Huntington III Project will furnish an additional update for the oral presentation at the 5th Annual National Educational Computing Conference.

At that time we plan to demonstrate sample courseware materials and to describe feedback from the 1982-83 trial of courseware materials. In addition, a preliminary edition of the Project's Handbook for Microcomputer Courseware Design will be available for inspection.

Questions from Teacher Evaluation Questionnaire

HUNTINGTON III: MICROCOMPUTER COURSEWARE DEVELOPMENT

Teacher _____ School _____

Grade level of users _____ Address _____

(Please use separate page for each grade level.)

_____ (Zip)

Name of class for which this courseware is being used _____

Name of courseware package _____

Please comment on or provide suggestions for each question.

1. Does the program meet the objectives stated in the teacher's guide?

   _____ Yes _____ No _____ Partly

2. Are the pre-programming activities effective in preparing the students to use the program?

   _____ Yes _____ No _____ Partly

3. Do the worksheets interface well with the computer program?

   _____ Yes _____ No _____ Partly

4. Are the instructions in the computer program clear?

   _____ Yes _____ No _____ Partly

5. Does the computer program highlight the important concepts?

   _____ Yes _____ No _____ Partly

6. Do students find the topic being studied to be related to the real world?

   _____ Yes _____ No _____ Partly _____ Does not apply

7. Does the program actively involve students in the learning process?

   _____ Yes _____ No _____ Partly

8. Did your students gain knowledge of applications of math/science concepts as a result of using the program?

   _____ Yes _____ No _____ Partly _____ Does not apply

(continued)

Teacher Evaluation Questionnaire (continued)

9. Are the graphics clear, relevant, and pleasing?

_____ Yes _____ No _____ Partly

10. Is the program easy to modify? (answer if appropriate)

_____ Yes _____ No _____ Partly

11. How much time did you spend in preparation for using this courseware package?

_____

12. Do you consider the amount of required preparation time to be reasonable?

_____ Yes _____ No _____ Partly

13. Do you have other suggestions for improving the courseware package?

_____

_____

_____

_____

Questions from Student Evaluation Questionnaire

HUNTINGTON III: MICROCOMPUTER COURSEWARE DEVELOPMENT

Name of computer courseware _____

Your grade level _____

Name of class for which you are doing this program _____

1. Did the discussion presented before you used the program prepare you adequately?

   _____ Yes _____ No _____ Partly

2. Were the worksheets coordinated with the program?

   _____ Yes _____ No _____ Partly _____ Does not apply

3. Did the instructions in the program enable you to use it effectively?

   _____ Yes _____ No _____ Partly

4. Did the program help you to understand applications of math/science concepts?

   _____ Yes _____ No _____ Partly _____ Does not apply

5. Did you find that parts of the program moved too slowly or too fast?

   _____ Yes _____ No _____ Partly

6. Did you choose to repeat parts of the program?

   _____ Yes _____ No _____ Partly Which parts? _____

   _____

   Why?--Fun? _____ More information? _____ Other? _____

   Explain _____

7. Does the material in the program coordinate with the subject matter you are studying?

   _____ Yes _____ No _____ Partly

8. Would you like to use more programs on different subjects?

   _____ Yes _____ No _____ Partly

9. How does the subject matter in the program relate to out-of-school experiences?

   _____

10. Do you have other suggestions for improving the courseware package?

    _____

    _____

A Universal Computer Aided Instruction System *

Henry Gordon Dietz
Ronald J Juels

Polytechnic Institute Of New York
Route 110, Farmingdale NY 11735

PILE, the Polytechnic Instructional Language for Educators, is a consistent universal language designed to provide a simple, yet versatile, language that non-computer-oriented teachers can use to prepare computer aided instruction (CAI) lessons. Developed under a grant from the Initial Teaching Alphabet Foundation, PILE supports the Initial Teaching Alphabet (and other character sets), graphics, and audio cues; further, PILE is designed to operate efficiently in microcomputer systems currently available and to be portable across vastly different hardware and software systems. However, the most important aspect of PILE is that it encourages more sophisticated use by hiding details of implementation, unlike Pilot which is simple by excluding abilities that do not have obvious implementations. This paper discusses the techniques used in PILE to achieve simplicity without sacrificing ability or portability.

Introduction

The current dearth of quality CAI software is well known. This unhappy situation parallels general software availability prior to the existence of modern software practices. Computer Scientists now recognize the importance of language definition, modularity, reuseability, and portability. PILE, a universal modern system for CAI software development, has been designed to address these important attributes. PILE was developed by researchers at the Polytechnic Institute of New York under a grant from the Initial Teaching Alphabet Foundation (ITAF) and will soon be distributed worldwide by the ITAF.

The system as implemented is portable to a wide variety of microcomputers, minicomputers, and mainframes, and encourages reuse of program modules. Educators with minimal experience in computer technology can easily learn to use the facilities made available by PILE and can incorporate modules as needed. While the cost of hardware is dropping dramatically, the requirement for rewriting programs already in place can greatly inhibit the widespread and cost effective use of CAI. The PILE system, designed for portability, can effectively operate across the spectrum of current and future hardware.

* This research project has been supported by a grant from the Initial Teaching Alphabet Foundation.

This paper discusses the design features and the embodiment of the PILE system.

Ease Of Expertise

Many CAI lessons are written as little more than the equivalent of a series of flashcards, hence it is easy to write such lessons in BASIC, Pilot, or nearly any other computer language. Unfortunately, more complex (hence more interesting and more effective) CAI lessons are difficult or impossible to write in most CAI languages unless the programmer is an expert. In defining PILE, the primary goal is to create a system which is almost effortless to master so that higher quality and reusable CAI lessons can be produced by educators who do not have an extensive background in computer science.

There are several major factors in design of a computer language system that is easy to master. These factors are: coherence of program structure with conceptual program structure [5], ability to build generalized software tools (modularity) [3], use of data objects (rather than types and rigidly specified structures) [1], and good readability inherent in the language (self-documenting code).

Program Structure: There is a strong tendency to adopt a trivial, inflexible, program structure for CAI lessons. Pilot, for example, is such a language; designed on the theory that trivial program structure is easier to learn. While, on the surface, it appears that such an approach would succeed, forcing a program with a different natural structure into a structure the language accepts shows the futility of this strategy. In Pilot, it is often impossible; in BASIC, it is possible only with great effort and an excellent understanding of the particular computer's implementation of BASIC.

There are two separate aspects of the structure of a program: the control structures used for looping and conditional execution of parts of a program, and the overall structure of the program. Neither aspect is unique to CAI.

In the early days of computer programming, there were many different theories concerning program structure. There remain a variety of approaches to overall structure, but Pascal-like control structures have become somewhat standard.

The advantage of a Pascal-like set of control structures is simply that people tend to think in terms of them naturally. Structures like if...then...else, while...do, and repeat...until are all common ways of expressing, in natural language, the concepts they represent. Since the semantics of the control structure mirror the natural language conception, fewer mistakes are made and more complex programs can be written.

In PILE, we have chosen to adopt a set of control structures as similar as possible to those in Pascal (although Pascal programmers will notice that PILE is more tolerant of semicolon and missing keyword errors). However obvious this choice may seem, these control structures are unusual in a CAI language.

The overall structure of a PILE program is also unusual for a CAI language, but consistent with the state of the art in computer language design. Each program consists of a main program and, optionally, functions.

The most difficult concept for the programmer using Pascal-like functions is nested scoping, so PILE has only two scopes: local to this invocation of a function or global (actually local to the main program). Functions may be defined anywhere except inside another function and may be invoked from anywhere (as in the C language). Recursion is supported.

No distinction is made between functions and subroutines (the returned value is ignored when called as a subroutine) and each invocation of a function can pass a different number of arguments, using call by value and referencing by position of arguments in the function call.

Software Tools: The easiest way to write a program is to reuse a program that is already written to perform that function. If parts of a program are designed as separate modules, or functions, then only functions and control structure that have not previously been written must be implemented; the majority of functions within most programs can simply be reused. There is nothing special about CAI that would make this general rule of computer science invalid.

In PILE, the implementation of functions makes reuse exceptionally uncomplicated.

Part of the simplicity of modular design in PILE is due to the generality of the arguments to a function. For example, in most languages a function which would return the maximum of a group of numbers could not be written to accept a variable number of arguments, so max(a,b) and max(a,b,c) would require two different max functions to be defined; a single max function can easily be written in PILE as:

```
function "max" begin
        biggest = arg1;
        while (argcnt # "0") do begin
                if (greater($("arg" ! argcnt),
                                biggest)) then
                        biggest = $("arg" ! argcnt);
                argcnt = argcnt - "1";
                end;
        return(biggest);
        end;
```

The other key feature of PILE in building modular software tools is dynamic linking at run-time. Since PILE is commonly run on microcomputer systems with modest resources, avoiding the overhead of having multiple copies of compiled functions is important.

However, dynamic linking in PILE does much more than save disk space. In PILE, the compiled code for each function is merely the value associated with the name of the function; functions can be manipulated as data. The overall effect is that functions exist in a huge virtual name-space (rather than in small main memory), updates of a function inherently update all references to that function. Modular use of functions does not require special libraries nor linking, and function names can be derived at run-time. A function need exist only if it is actually called (useful in testing programs before all the functions have been written).

(Self-modifying code and programs that create and then execute functions within themselves are possible, however, such techniques are rarely employed, and there are safeguards to insure against accidentally executing data.)

Data Objects: Until computers and humans begin naturally communicating in the same language, there will always be a trade-off between native language for computer and programmer. Making the computer more efficient is easier than making the human more efficient, therefore, PILE opts for a form most natural to humans. Data types, and data structuring, are both at the center of this trade-off.

Traditional CAI languages standardize on a single data type, but no data structures, which limits the ability of the system to an unacceptable level.

Traditional computer science imposes a wide variety of data types and structures. Pascal, C, and PL/1 have approximately half their complexity (measured by the BNF productions) dedicated to data typing and declarations. There is no need for this complexity other than to increase the execution efficiency. However, CAI programs are rarely limited by computational speed, even on microcomputers.

The AI (Artificial Intelligence) community has developed several languages that effectively use one universal data type and one universal data structure. Lisp uses atoms and lists, Logo uses objects and lists, and Snobol uses strings and tables; the appelations differ far more than the implementations [4]. In much the same way, PILE uses values and environments.

A value is really nothing more than a variable-length character string. Variables in other languages are names in PILE. An environment is merely a list of name-value pairs considered as a whole. Within a non-local environment, a name is referenced as "environment"["name"], even if the environment is stored in a file on disk. There are no data declarations.

Operations on data are also similar to those found in Lisp and Snobol, except that PILE uses conventional algebraic notation for all expressions.

Powerful, Flexible, Features For CAI

Thus far, the features discussed have been applicable to general-purpose computer languages as well as to CAI systems. To this extent, PILE is a general-purpose language. However, there are special abilities that are so commonly used in CAI that, for reasons of efficiency, they ought to be directly embedded in the language. These features involve database-like referencing, character fonts, graphics, and audio cues.

Database-like Referencing: PILE environments make relational database operations simple (although perhaps wasteful of disk space). For example, the answer given by a student can be entered in a database (on disk) by:

```
"question"[student_name] = student_answer;
```

Although PILE also supports conventional disk I/O, when using the array-like referencing of environments, the simple assignment in the above example is all that is needed... there is no open file, close file, nor statements to scan for the student's name; it is all transparent to the programmer. Reading is equally simple. The only disadvantage is that there must be one environment for each relation expressed in the database.

Character Fonts: The Initial Teaching Alphabet (ITA) uses a set of phonetic characters, in addition to the traditional lower-case alphabet, to aid in teaching written English. Since PILE was developed for the Initial Teaching Alphabet Foundation, the system must efficiently deal with non-standard (relative to computers) fonts.

In order to provide flexibility, PILE supports the ITA, and user-defined character sets, using stroke-fonts. A stroke-font consists of a list of character definitions. For each character, the proportional spacing width, the sequence of pen strokes to draw it, and the 8-bit code to represent it internally are specified.

Using this implementation, PILE programs can control the way in which text is written to the display by choice of stroke-font, character size scaling, and stroke rate (the speed with which the pen-strokes of each character are drawn). Still, text is written to the screen by a simple Pascal-like write or writeln.

Graphics: Graphics used in CAI systems tend to be simple drawings with little or no animation. PILE graphics are designed to make drawings, at that level, easy to create and to transport to different displays.

To accomplish this, the PILE system includes an image editor. The image editor writes a program which results in the drawing when executed by the PILE Interpreter (PILEI). Since the image is specified as a mathematically-precise entity, although it is drawn in a conventional manner, PILE programs can adjust the image to the characteristics of different displays.

(Turtle graphics are also supported, but are not embedded in the language.)

Audio Cues: As many other CAI systems, PILE supports control of a cassette recorder. However, PILE also supports phonetic voice synthesis. Each font can have different pronunciation rules for phonetic translation, and intelligible speech can be produced from text. Text in the ITA font is pronounced particularly well due to the phonetic nature of the ITA. In addition, natural languages which have no written form can be supported. Regardless of font, pronouncing a phrase is as simple as:

```
say(phrase);
```

Portability

There is no standard computer. Hence, for purposes of portability, PILE defines its own standard which is easily emulated on currently available systems.

Pseudocode: PILE is implemented as a compiler (PILEC) that generates an internal pseudocode, an interpreter (PILEI) that executes this pseudocode, and a series of utilities to aid in building lessons written in PILE. Unlike the pseudocode implementations of Pascal, the speed of execution of PILE programs is not significantly degraded by the overhead of interpreting the pseudocode; like Snobol, the typical PILE pseudocode instruction takes much longer to execute than to decode.

The entire PILE system, including the pseudocode interpreter (PILEI), is written in C. This permits porting to most microcomputers and minicomputers. While an assembly language version of PILE would be faster, most of the speed increase is easily achieved by re-writing only a few C functions in assembly language. On a 4Mz Z80a, approximate relative speeds are: 1 for PILEI in pure assembly language, 1.5 for PILEI in C with 5 simple functions re-written in assembler, and 3 for PILEI in pure C.

In addition to the efficiency of generated code, C is the language of choice because of its popularity and the quality of the standard definition of C. While BASIC and Pascal have many mutant versions, C remains relatively pure.

Internal Data Formats: Just as the PILE system and compiled PILE programs must be portable, all other forms of data used by the system must be hardware and (operating system) software independent. To simplify this, there is only one form used for disk storage: the file format for PILE environments. Compiled PILE programs, data, images, configuration files; all files used by the system are in this same format.

Wherever possible, data is formatted in such a way as to be crudely human-readable. Since these files must be able to be transferred from one machine to another, using printable ASCII for data files usually is more convenient.

Graphics Meta-forms: The largest obstacle to portable CAI lessons appears to be portable graphics. No two displays are even remotely similar. The only solution is to abstract a graphic meta-form which can be applied to nearly all displays.

In PILE, this graphic meta-form consists of points, vectors (lines), ellipses, and filled areas (seed fill). These constructs apply to any device (although filled areas are not implemented on some displays), and are used for character stroke-fonts as well as graphic images. PILE graphics have been successfully ported to Apple IIs, NorthStar Advantages, Tektronix 4006 terminals, HiPlot DMP4 plotters, and Zenith Z19 terminals.

Conclusion

In this paper, a system for developement of sophisticated CAI software has been presented. This system is very complex; however, this complexity is not experienced by the typical user of the system. By burying most of the complexity of CAI within the system, the CAI author is freed from most programming concerns.

The key to writing an effective program is the ability to make the program do what is desired. Although writing a trivial first program in PILE might seem awkward, writing a complex program is much simpler in PILE because the system is compatible with modern programming practices and provides many tools. According to the intermediate COCOMO model [2], a significant software project requires only 44% of the effort required for the same project without these techniques. We expect that these claims will be supported by a large user community shortly after PILE's first general release later this year.

References

[1] Harold Abelson, A Beginner's Guide to Logo, BYTE, Volume 7, Number 8 (August 1982), pages 88-112.

[2] Barry W. Boehm, Software Engineering Economics, Copyright 1981, Prentice-Hall, pages 114-144.

[3] B. W. Kernighan and P. J. Plauger, Software Tools, Copyright 1976, Addison-Wesley Publishing Company.

[4] T. W. Pratt, Programming Languages: Design and Implementation, Copyright 1975, Prentice-Hall.

[5] N. Wirth, Foreword to A Primer On Pascal, Copyright 1976, Winthrop Publishers, pages XI-XII

A STUDY OF STUDENT-COMPUTER INTERACTIVITY

David Trowbridge
Educational Technology Center
University of California, Irvine, CA

Robin Durnin
Claremont Graduate School, Claremont, CA

Abstract

A research project being conducted at the Educational Technology Center is addressing questions about interaction among individuals and small groups as they use computer based learning materials. A videotaping system that utilizes an interface device between a microcomputer and a videotape recorder is described. An observational instrument is presented for analyzing the interactions among group members and the computer program. Results of a pilot study and future directions of a formal study are summarized.

Introduction

The primary advantage of computers in education may well be the high level of interactivity that properly designed learning materials can provide to students. Educational research has indicated that active involvement of the learner is an essential condition for the development of reasoning skills, the formation of concepts and the acquisition of problem solving skills (Hilgard, 1975). We would expect that learning environments which provide greater opportunity for active engagement would be more effective than those which provide less. Certainly, the computer has the potential for providing a highly interactive environment.

This project uses a collection of highly interactive computer dialogs that were developed under two previously funded projects at the Center: Science Literacy in Informal Learning Environments, and Formal Reasoning Skills for Young Adolescents. The learning modules we have chosen for this project have been described elsewhere (Bork, 1981; Trowbridge, 1981). They are Batteries and Bulbs, Speed, Optics, Tribble Families and Sherlock Holmes. They all use extensive graphics to simulate experimental situations and place the student in the role of a scientist or investigator. The simulations are embedded in Socratic dialogs. Students play the role of experimenters, manipulating objects on the screen, and engage in a dialog with the program about the experiments. Both the simulated experiment and the dialog require frequent keyboard activity.

The learning materials developed at the Center have been field tested in junior high schools, science museums and libraries and in various social settings. We have found a high level of interactivity among students using these dialogs, whether one student is working alone at the keyboard, or several are working together.

Frequently, when there are more than one student at the computer, conversation is lively; group members talk with each other often, offering assistance, encouragement, opinion and argument. The nature of the interaction in this environment is diverse. A given individual interacts both with the computer program and with the other group members. Primary interaction with the program is via the screen and the keyboard. Interaction with other individuals has both cognitive and social functions. Communication is both verbal and non-verbal. Social interaction is sometimes cooperative, and sometimes competitive.

Pilot Study

In the initial stages of this investigation, we have attempted to identify variables of interest, choose a research strategy, develop a system for data collection, and construct and validate an observational instrument. These aims of the pilot study have been met and are reported in this paper. In the formal study, we will seek to test certain hypotheses arising from the earlier work. There, we will investigate

the effects of group size on selected educational outcomes: interactivity, frequency of success and achievement. Later in this paper, we will outline the directions of the formal study.

Thirty-five students in grades 6-8 took part in the pilot study. They worked at the computer as individuals or in groups of two or three. In consultation with their teacher, we selected groups representing a wide cross-section of students, of high, medium and low ability. Some groups had worked together before, others had not. Some groups were chosen to be heterogeneous with respect to ability levels, others homogeneous. Some groups were composed of individuals of mixed sex, others of the same sex. We also used a variety of computer based learning materials which varied in several respects, such as in how much graphics they used. Thus, the results of the pilot study are drawn from a fairly diverse collection of groups and computer materials.

Research Strategy

We have chosen to use a method of interactional analysis, coupled with a videotaping system for investigating interaction in this special environment of groups and computer based learning materials. We have attempted to operationalize the idea of interactivity by defining a quantitative measure of the level of interactivity that can provide an indicator of the degree to which a particular student is actively involved in the learning session. We use this to examine the effect of group size on the level of interactivity.

Videotaping System

As students use the computer based learning materials, the sessions are recorded on videotape. Three components of the group activity are recorded: (1) video of the students working together, (2) audio of their conversation, and (3) all key pushes on the computer keyboard. The videotape data collection system consists of a microcomputer that runs the interactive learning materials, a video cassette recorder with two separate audio channels, an interface device for connecting the computer to the recorder, and associated utility software for handling input and output during recording and playback (Figures 1, 2).

Verbalizations are recorded on one of the audio tracks of the cassette tape, and keystrokes are recorded on the other. During playback, the keystroke codes are used to drive the computer in synchronization with the video and audio components. The computer programs have been modified so that during the recording session, all keystrokes are transmitted to the videotape recorder. During playback, the keyboard is disabled and all keystrokes are received from the videotape player. This underlying software is a modification of the TextPort and GraphPort systems developed under UCSD Pascal in earlier projects at the Center.

Taping Sessions

Students were brought from their school to the university at the end of the school day. They worked alone or in groups of 2, 3, or 4, spending up to an hour at the computer. The television camera was in full view, and students were told that they were being filmed. Usually, they became quickly engrossed in the computer dialogs and paid little attention to the fact that they were being recorded. No adults were present in the room during the learning session; consequently, students were generally expressive and uninhibited.

Observational Instrument

Our observational instrument has been derived from the methods of interactional analysis developed by Bales (1950) and Flanders (1970). Bales' system for observing small groups engaged in problem solving situations has twelve categories (e.g., Gives suggestion, Asks for suggestion, etc.) grouped into two social-emotional areas (Positive and Negative) and one task area (Neutral). Flanders' system for analyzing teacher behavior also has multiple categories, most of which relate to teacher talk in the classroom.

Our own instrument for interactional analysis is suitable for recording both verbal and non-verbal behavior. Use of videotapes allows us to categorize gestures, actions and facial expressions which do not involve vocalizations. The system of categories we have developed in the pilot study is shown in Table 1.

Table 1. Interactive Behavior Codes

	Keyboard Interaction
K	types at keyboard
	Verbal-Cognitive Interaction
T	tells, directs others
Q	queries, asks for suggestions
A	accepts, responds to suggestions of others
L	looks away to ponder or discuss with others
I	interprets in one's own words
X	explains, formulates reasons
M	formulates question or answer
P	formulates prediction
E	evaluates using criteria
D	disagrees with program or objects to message
	Cooperative-Social Interaction
n	neutral conversation, opinions
a	approval, agrees with another
d	disapproval, disagrees
s	shares keyboard with others
t	takes turns
h	gives help, assists another (aid to action)
v	polls others, solicits, votes
y	delegates task to another
e	encourages another

Coding Procedures

Observers view two screens during playback of the videotapes: the computer screen as it appeared to students using the learning material and a television screen showing the students themselves. During the playback session, observers have a coding form before them with a sequence of boxes corresponding to each ten-second time interval during the session. They use the digital timer on an auto search controller to keep track of the frame numbers on the video. In each ten-second period, each observer makes one or more single-letter entries (typically, no more than 4 observations during each 10-second interval) in a box.

1. Interaction with Computer

While it can be argued that reading from the computer screen is a form of interaction with the program, we have chosen not to include this behavior in our measure of interactivity. We have found that for group sizes of 1, 2 or 3, each member typically watches the computer screen more than 95% of the time. It is a relatively passive activity, akin to reading from the page of a textbook or listening to a lecture. It usually indicates attentiveness to the learning activity, but it is not active in the sense that typing at the keyboard is.

Our category, K, for keyboard activity was used to record instances of keyboard input excluding simply pressing the space bar to continue (a standard convention used in these programs to ensure that messages are not cleared from the screen before the user is ready to go on). We have included all cases of using the arrow keys to move the graphics cursor, pressing keys for single character input and typing in words and sentences. With the elimination of simple "page-turning" input, the category, K, is counted as active participation on the part of the student.

2. Verbal-Cognitive Interaction

Ten categories of behavior represent instances in which the subject is apparently engaged in some kind of cognitive activity related to the content of the learning material. Most of these categories are verbalizations concerning the program. Verbalizations that were not related to the program in any way were categorized as Off-Task. Other verbalizations that were not cognitive could be identified as Cooperative-Social.

3. Cooperative-Social Interaction

Nine categories of behavior represent socially supportive and cooperative forms of interaction. While we did record a few categories of anti-social behavior (e.g., wresting the keyboard forcibly from another individual, verbally abusing another person, or consciously ignoring the talk or other behavior of another person), this happened relatively infrequently. Competitive behavior was also relatively rare. The behaviors labelled Cooperative-Social are all

considered to be generally beneficial to the learning process. Any social behaviors that were distracting to the learning activity were categorized as Off-Task.

Measures of Interactivity

To quantify the observations of interactive behaviors, we define Interactivity, Interactivity Rate, and three components of Fractions of Interactivity Rate.

A numerical value for Interactivity of a particular individual participating in a given session may be defined as the total number of interactive behaviors observed for that session according to our table of behavior codes. By dividing the Interactivity by the corresponding number of 10-second coding intervals over which the observations were made, we obtain an Interactivity Rate. The Interactivity Rate represents the degree of active involvement of an individual participant in the learning activity.

We also define three components of Interactivity Rate corresponding to the categories Keyboard, Verbal-Cognitive and Cooperative-Social. To determine the Fractions of Interactivity Rate in a particular learning session, we count only the number of observations falling within a particular category and divide by the number of 10-second intervals.

Results from Pilot Study

In this section we present results from a sample of sixteen sessions. Six involved the first four activities of the Batteries and Bulbs program (Bork, 1981). Five sessions involved the first three activities of the Speed program and five sessions involved the final two activities (Trowbridge, 1981). Four sessions consisted of 3 students working together at the computer, seven sessions had 2 students, and five sessions had 1 student working alone (Table 2). Time to complete the four activities ranged from 25 to 45 minutes. Students typically remained attentive to the lesson for at least 30 minutes. Overall, interactive behaviors occurred an average of once every 20 seconds.

Table 2

Interactivity Rate by Group Size

Subject Matter	1	2	3
Elec 1-4	.41(2)	.57(2)	.43(2)
Speed 1-3	.45(2)	.82(3)	
Speed 4-5	.24(1)	.45(2)	.49(2)

Group Size

1. Interactivity Rate as a function of group size

We would expect that an individual working alone would have a lower rate of interactivity than when working as a member of a pair, simply because when working alone, modes of verbal-cognitive and cooperative-social behavior are not available. Furthermore, we would expect that if the group size were to grow very large, then the Interactivity Rate for any particular individual would fall. Presumably, a small group with some number of members greater than one would result in the greatest interactivity. Observations on this small sample is consistent with this expectation (Table 3).

On several occasions, we have observed that in groups of three, one person tended to withdraw from the group's activities. In order to test whether this is a likely occurrence, we have categorized the members of each 3-person group on the basis of the Interactivity Rate of each into the High, Middle and Low member of the group. Then we have plotted the averages for the High person's Rate from each group, the Middle person's Rate and the Low person's Rate (Figure 3). A conclusive generalization will have to wait until we have obtained observations from a larger number of groups. However, if the graph retains the shape suggested by these few data, we shall conclude that for groups of three, there is a tendency for one member to be left out.

Table 3

Interactivity Rates

	Group Size 1	2		3		
		low	high	low	med	high
I'_k	.21	.17	.20	.13	.13	.14
I'_c	.15	.21	.36	.08	.21	.28
I'_s	.00	.14	.17	.07	.14	.14
I'	.36 (5)	.52 (8)		.73 (8)	.28 (3)	.48 (4) .59 (4)

Note:

(1) Combined results from Elec 1-4, Speed 1-3, and Speed 4-5

(2) (#) indicate the number of sessions analyzed

2. Mode of Interactivity as a function of group size

Three modes of interactivity may be separated from the overall Interactivity Rates described above: Keyboard, Cognitive and Social. As expected, the fraction of Keyboard Interactivity decreases as the size of the group increases (Figure 4). In addition, both Verbal-Cognitive and Cooperative-Social Fractions are highest for groups of two (Figures 5, 6). The two individuals who worked alone were instructed to express their thoughts verbally as they used the computer materials; thus we see a significant portion of their interaction as verbal-cognitive. In the formal study, we hope to ascertain whether the observation of maximum Interactivity Rate for groups of two is statistically significant.

Directions for the Formal Study

As a result of the pilot study, we decided that the variable of group size was worth examining in greater detail, and this became the focus of subsequent research. In addition, prior group history seemed to have an effect on interaction, so this became a secondary variable of interest. The mixture of ability levels also seemed to have an important role, but due to limited resources, we chose to control for this variable by selecting all of our subjects from comparable, middle ability students for the formal study. We have not found sex to have a strong effect on interaction, so have chosen to combine single and mixed sex groups. For the formal study we are using the same instructional materials with all students (Batteries and Bulbs).

We are examining the effects of both group size and prior group experience on interactivity and achievement. Interactivity is measured using the observational instrument developed in the pilot study. Achievement is measured using a pencil and paper test which is administered immediately after the session with the computer. The post-test consists of eight questions, each showing an arrangement of batteries, bulbs and wires and asking whether the bulb would light in each case.

We have videotaped fifty-six 7th and 8th grade students in 26 sessions with the computer. They were generally college-bound, of middle ability students, from fairly affluent middle class backgrounds. Taping sessions took place in the morning, with an adult present in the room in a non-supervisory capacity.

Table 4. Sample for the formal study

		Group Size Ind.	2	3	4
	Numbers of Groups				
Cooperative Experience?	Yes	5	5	4	2
	No	3	3	2	2
	Numbers of Students	8	16	18	16

The hypotheses we hope to test in the formal study are:

1. Students working in pairs have higher measures of interactivity than students working in other grouping arrangements.

2. Interactivity measures for acquainted student groupings are higher than for unacquainted groupings.

3. Post-session achievement measures for students who have worked in pairs are higher than for students working in other grouping arrangements.

Conclusions

The videotaping system described here, which couples the computer to a video recorder, provides a rich source of information about the use of computer based learning materials. We have only begun to explore the possible applications of a system that can reproduce human-computer interactions at a detailed level.

The behaviors coding scheme we have described enables us to quantify the overall level and quality of interactivity in this learning environment. Preliminary evidence suggests that students working in pairs have the greatest opportunity for high interactivity. Soon, we expect to be able to make some generalizations about how various kinds of computer based learning materials affect the interactivity of the learner as well.

Results of investigations such as this should provide guidance to developers of computer based learning materials as well as to classroom teachers who must manage the logistics of computer usage in their classrooms

Acknowledgements

The interface device was designed and built by Michael Potter. Associated software was developed by John McNelly and Steven Bartlett. Elinor Coleman and Cynthia Powell have helped in the coding of the videotapes. The authors wish to thank each of these peole for their contributions to this project.

References

1. Bales, R., 1950. Interaction Process Analysis: A Method for the Study of Small Groups, University of Chicago.

2. Bork, A., Kurtz, B., et al., 1981. "Science Literacy in the Public Library - Batteries and Bulbs," Proceedings of the National Educational Computing Conference.

3. Flanders, N., 1970. Analyzing Teaching Behaviors, Addison-Wesley Publishing Co.: Menlo Park.

4. Hilgard, E. R. and Bower, G. H., 1975. Theories of Learning, 4th Ed., Prentice-Hall: Englewood Cliffs, New Jersey.

5. Trowbridge, D. and Bork, A., 1981. "Computer Based Learning Modules for Early Adolescence," Computers in Education, R. Lewis and D. Tagg (editors), North-Holland Publishing Company.

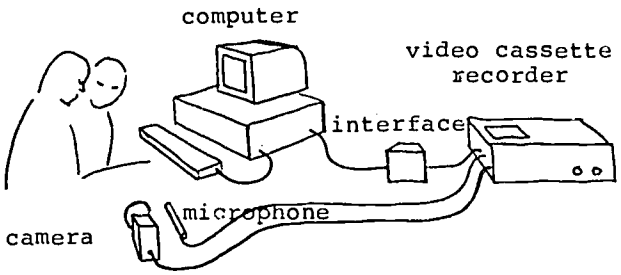


Figure 1. Setup for recording sessions

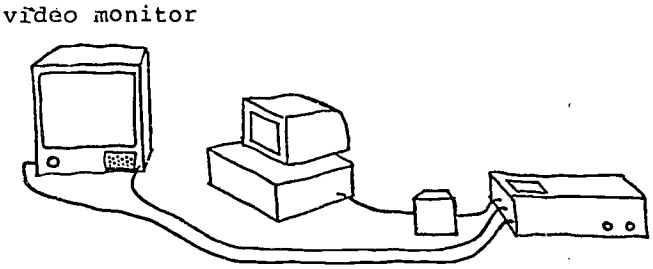


Figure 2. Setup for playback sessions

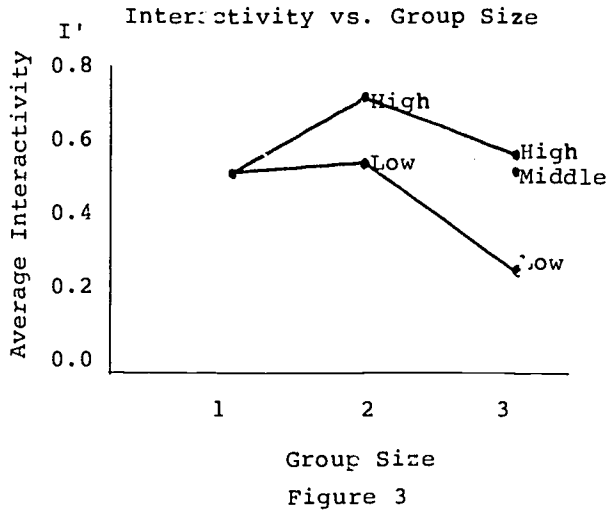


Figure 3

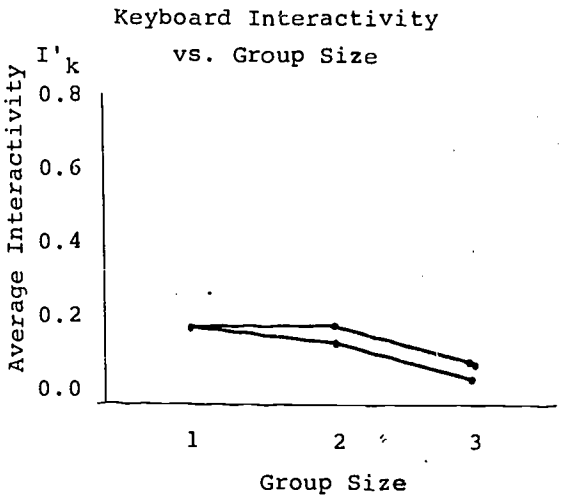


Figure 4

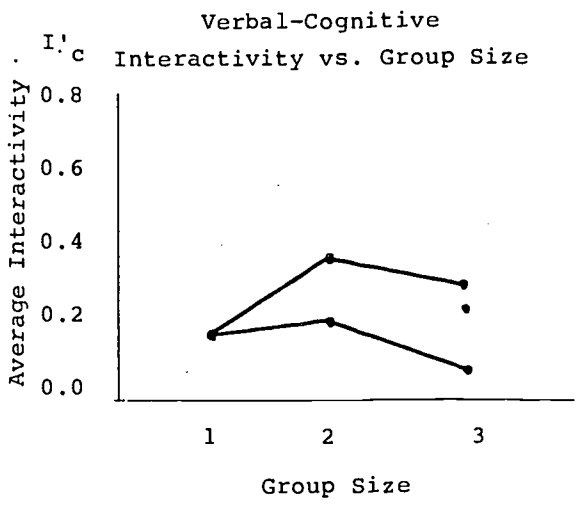


Figure 5

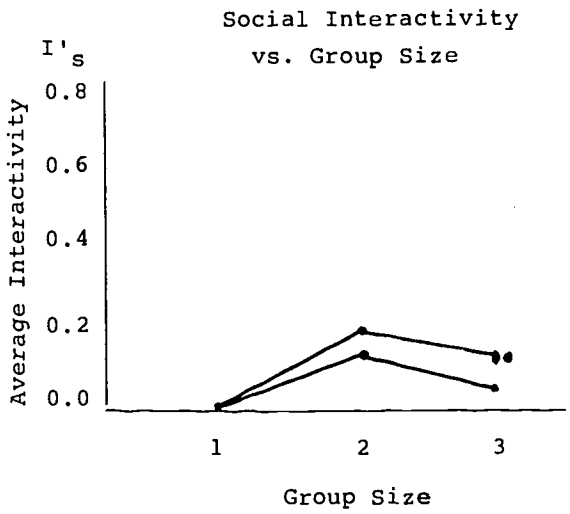


Figure 6

THE IMPLEMENTATION OF TECHNOLOGY AND THE CONCERNS-BASED ADOPTION MODEL

By Dr. Cheryl A. Anderson

Department of Curriculum and Instruction
University of Texas at Austin

Abstract

This paper presents a brief synopsis of the literature on the change and adoption process as it relates to the task of infusing computer technology into the school system. The Concerns-Based Adoption Model (CBAM) is also discussed. The model is based on ten years of research on teacher behavior with educational innovations. According to the author, CBAM can provide a theoretical framework for planning a computer implementation effort. The model provides the change agent with a set of tools for measuring the success of the adoption; for measuring teacher attitudes and feelings toward the innovation; and for measuring teacher behaviors with the innovation. By using these tools, a change agent is assured of using a more systematic approach to the implementation effort.

It seems that schools are anxious to jump on the computer technology bandwagon. Many people are concerned about the potential for failure, in part because similar attempts to change established school practices have failed. A long line of innovations has preceded the computer, including instructional television, programmed instruction, teaching machines, team teaching, individualized learning, and modern math. Few of these made their way permanently into the classroom. Why should we expect the computer to be accepted with enthusiasm when so many innovations have failed? Indeed, Oettinger (1969) suggests that computer technology will not be effectively used because its very nature works against the rigidity of the school system. Karen Sheingold (1981), in a study of the implementation issues relating to computers in the schools, states: "While the new technology does differ in many important respects from the old, expectations about educational impact must be viewed cautiously. There are many steps between putting a machine, albeit a powerful, engaging machine, into a classroom and making a difference for children and teachers" (p. 4).

For those agents of change who are responsible for seeing that computer technology is successfully infused into the school system, there is a body of literature on the change and adoption process which may provide some guidance. There is also a change model called the Concerns-Based Adoption Model (CBAM) which relates specifically to the teacher and the adoption process. The purpose of this paper is to present a brief synopsis of the relevant change literature and to provide an introduction to this model. First, however, it is necessary to look at why the existing school system is so difficult to change.

Current School Model

Part of the problem in trying to implement a new technology, such as computers, is that the current school system is designed around the traditional teacher-learning model (Pitts & Schneider, 1981). The model is teacher- and textbook-dominated. Computer technology challenges this traditional focus because, instead of information that comes from the teacher or the text, the student's work revolves around the computer. With computer technology, teachers' roles will also change (Sheingold, 1981). Unfortunately, schools are very stable institutions that are resistant to any kind of change (Goodlad, 1976). Not only is the system as a whole resistant, but the individuals within the system are resistant as well. The change process is very dependent upon the individual, particularly in the educational system. This is because the individual teacher assumes a degree of ownership over an innovation during the process of assessing its benefits and testing its value (Podemski, 1980). This concept of ownership is critical in education because the individual teacher is allowed a certain amount of autonomy with regard to the curriculum materials and procedures he or she uses in the classroom (Podemski, 1980). It is often the case that the individual teacher's attitude toward an innovation determines not only how it is used, but whether it is used (Hall, Wallace & Dossett, 1973).

The Individual and Change

That the individual is the primary focus of efforts at technological change is evident in social science literature. Wolcott (1981) notes that social scientists have much to offer the change agent who is trying to implement a

technology. He summarizes some of the problems which have been uncovered through studies of human behavior and change. These problems include the following:

1. human will resist any change that threatens their basic security;
2. humans have a tendency to resist any technology that they do not understand;
3. humans will not vary their behavior unless they can use the new technology to satisfy a need that is not being thoroughly met by existing technology;
4. humans cannot be forced to accept technological change;
5. in order to be accepted, an innovation--particularly one that originates from extensive knowledge of science and technology--must be made intelligible and be given a place of value by the adopting culture; and
6. resistance to change is often centered around the way in which the innovation is administered rather than the innovation itself.

Wolcott (1981) reminds developers and disseminators of technologies ". . . that lessons about change go unheeded only at considerable risk" (p. 25). He also points out that the typical teacher rarely values innovations because most do not address the teacher's real problems or provide real help in the classroom. Thus, if the computer is to be implemented successfully, it is the individual teacher who must be won over. The plan of action must present the computer in a non-threatening, understandable manner in addition to providing uses which will be valued by and of interest to the teacher.

Concerns-Based Adoption Model

As mentioned previously, CBAM could provide a useful framework for those who are responsible for implementing computer technology in the schools. This model is based on ten years of research by the Research and Development Center for Teacher Education at the University of Texas. The model is based on the following assumptions:

1. that change is a process, not an event; neither a principal's edict nor a one-day workshop can assure change, because change takes time;
2. that individuals within the institution must change before the institution can change--thus the individual is the primary focus of the CBAM research;
3. that the individual's personal feelings, perceptions, and motivations concerning an innovation play an important role in determining the success or failure of an innovation;
4. that change is developmental; that is, an individual will move through identifiable stages of concern about the innovation and levels of skill in using the innovation during the change process;
5. that an innovation is being implemented; it can be a curriculum material, a technology, or a teaching method that is new to the school;
6. that the innovation is appropriate to the system and has the potential to be effective within the school system; and
7. that there is an informal or formal leader who represents the change agent for the process. The function of this person is to design the effort in an adaptive and systematic way so that the state of change is assessed and reassessed (Hall, 1979). This allows the change agent to select certain interventions based on the latest diagnostic data. Thus, effective training in the use of an innovation occurs when the trainer uses a diagnostic and prescriptive model so that training can be targeted to individual needs.

Within the model, several diagnostic tools have been developed to assess where an individual is in relationship to the training effort. Three diagnostic tools have been developed: Stages of Concern, Levels of Use, and Innovation Configurations. What follows is a brief description of these concepts.

Stages of Concern

Stages of Concern about an Innovation are the feelings, motivations, and perceptions that individuals have as they progress through the adoption process. The CBAM researchers have identified seven stages:

1. awareness: at this stage there is little involvement with or concern about the innovation;
2. informational: here the individual is only interested in receiving general information about the innovation and he or she is not thinking about how the innovation will effect him or her personally;
3. personal: at this stage the individual is concerned about the demands that the innovation is making on his/her time; consideration is given to the personal benefits of using the innovation as well as the potential perils;
4. management: the individual focuses on the processes and tasks of using the innovation; concerns deal with the best use of resources, management of time, efficient use of the innovation, and organizing use;
5. consequences: at this stage the individual begins to consider the impact that the innovation will have on the students; the concerns focus on the relevance of the innovation to students and on the outcomes based on student performance;
6. collaboration: the individual has the desire to cooperate with other people who are also using the innovation; and

7. refocusing: this is when the individual begins to express concerns about replacing or modifying his or her use of the innovation (Hall, 1979).

Knowing a teacher's Stages of Concern can be of use to the change agent in planning in-service training. The CBAM research indicates that, for training to be effective, there must be a match between the personal concerns, the expertise of the trainee, and the method of training used to facilitate the use of an innovation (Hall, 1978). For example, nonusers of computer technology would be most concerned with obtaining some general information about the innovation and exploring the implications that it has for them on a personal level. A workshop dealing with the consequences of using computers would be inappropriate at this time; however, a simple hands-on experience would meet their needs. CBAM research indicates that in the beginning of the implementation process, teachers are not concerned with the implications for students and will not be until their own personal concerns are overcome.

Three different procedures have been developed for assessing the stages: an interview technique, an open-ended question technique, and a 35-item questionnaire. By analyzing the stages through these methods, a profile can be obtained. This profile will typically show that some concerns are more intense. This intensity should progress to different stages. This enables the trainer to tailor the different interventions, whether they be workshops or individual guidance, to the profile. The Stages of Concern focus on the attitudes or feelings that an individual has toward an innovation. The next facet of CBAM focuses on the behaviors that the individual exhibits when attempting to use an innovation.

Levels of Use of an Innovation

The Levels of Use measurement focuses on eight behaviors which can be identified through observation and interview techniques. These behaviors include:

1. nonuse: the individual has no knowledge or involvement and is not attempting to change;
2. orientation: the individual makes a decision to seek information about the innovation;
3. preparation: the individual makes the decision to devote time to learn to use the innovation;
4. mechanical use: the individual is using the innovation in an uncoordinated manner and his or her behavior focuses on mastering the task of use of the innovation;
5. routine: the individual stabilizes his or her use, little time is required to prepare for use, no changes are made in how the innovation is used, and there is little thought about improving use;
6. refinement: the individual initiates efforts to increase student outcomes, and thought is given to both the short-term and long-term consequences for students;
7. integration: the individual initiates efforts to combine his or her use with the related activities of other teachers to obtain a collective impact on the students; and
8. renewal: the individual starts to re-evaluate the quality of the innovation and begins to make modifications or seeks other alternatives to the innovation (Hall, Louchs, Rutherford & Newlove, 1975).

Researchers at the R & D Center have discovered that, regardless of the type of innovation, the majority of the teachers stay at the level of routine use. In the first year of use 60-70% of the individuals will only reach the mechanical-use level. They have also discovered that the progression is not always a lock-step one; an individual can move out of sequence. However, the CBAM researchers have found that individuals do not collaborate until they have personally mastered the innovation.

Innovation Configuration

The third component of CBAM is Innovation Configurations (Hall & Louchs, 1978). It is based on research which indicates that innovations are often adapted or modified by individuals to suit their particular needs or situation. Often an innovation is so drastically modified that it is no longer used in the manner originally intended by the developer. Innovation Configurations reflect the different patterns that result from the selection and use of key elements or components of an innovation. These components could include: instructional objectives, materials, equipment, grouping patterns, and tests or some means of indicating that the innovation has been implemented. The purpose of identifying various patterns is to determine whether or not the implementation has been successful. The procedure suggested by the CBAM researchers provides the change agent with a checklist that can be used for this evaluation. The procedure is as follows:

1. conduct interviews with either the developer or the person who is responsible for facilitating the adoption of an innovation--the interviewer tries to identify key components by determining what the innovation will look like in relationship to the behaviors of the students and the teachers;
2. conduct interviews and observations with a small number of users to determine the possible variations of the components;
3. develop interview questions and interview a larger number of users--questions are asked about teaching practices with the innovation;
4. construct a checklist which contains the key components and the possible variations within each component; and
5. fill out a checklist on all users and determine what dominant patterns exist (Hall & Louchs, 1978).

The benefit of developing a checklist is that it can help clarify how the innovation is supposed to operate when it has been fully implemented. This model is important in evaluating the success of the implementation effort. Facilitators of change and in-service trainers can use this concept to identify the components that need to be targeted for further effort (Hall, 1978).

Conclusion

This has been a very brief introduction to the Concerns-Based Adoption Model and the change literature which supports its use. The model gives those of us interested in implementing computer technology a theoretical framework and a set of tools for measuring our success in our implementation efforts. It provides us with a method for systematically analyzing our efforts to facilitate change. By using the Stages of Concern, we can determine how teachers feel about using the computer in the classroom. By using the Levels of Use, we can determine how skilled they are at using the computer. And by using the Innovation Configuration process, we can first identify the key components needed for full implementation of computer technology in the classroom, and then evaluate our success in relationship to those key components. CBAM has been used by the Montgomery County Public Schools in Rockville, Maryland, in their efforts to implement computer-literacy training for K-6 teachers. School representatives state "Rooted in both theory and practice, CBAM includes strategies that may be used to guide teacher education as well as to provide a basis for determining when elements of curriculum have been implemented" (Phillipp, Muntner, & Cutlip, 1982, p. 321). There are valuable lessons to be learned by looking at this body of literature. If we choose to ignore these lessons, then it is likely that computers will join other technologies which have failed to impact teaching practices and which are relegated to the storage closet.

REFERENCES

Goodlad, J. I. "Schooling and education." In R. Hutchins (Ed.), The Great Ideas Today. New York: Encyclopedia Britannica, Inc., 1976.

Hall, G. E. Concerns-based inservice training: an overview of concepts, research and practice. Paper presented at Conference on School-Focused Inservice Training, March, 1978.

Hall, G. E. Using the individual and the innovation as a frame of reference for research on change. Paper presented at the Annual Meeting of Australian Association for Research in Education, Melbourne, November, 1979.

Hall, G. E. & Louchs, S. F. Innovation configurations: analyzing the adaptions of an innovation. Research and Development Center for Teacher Education, University of Texas at Austin, November, 1978.

Hall, G. E., Louchs, S. F., Rutherford, W. L., & Newlove, B. W. "Levels of use of the innovation: a framework for analyzing innovation adoption." Journal for Teacher Education 26(1), 1975.

Hall, G. E., Wallace, R., & Dossett, W. A Developmental Conceptualization of the Adoption Process Within Educational Institutions. Research and Development Center for Teacher Education, University of Texas at Austin, 1973.

Oettinger, A. G. Run, Computer, Run. Cambridge, Massachusetts, Harvard University Press, 1969.

Phillipp, C., Muntner, J. & Cutlip, P. Computer literacy for K-6 teachers. Paper presented at the 20th Annual Association for Educational Data Systems Conference, Orlando, Florida, May, 1982.

Pitts, M. & Schneider, J. Educational Technology: Bright Promise or Dim Future. CEDAR Proceedings of a Cooperative School Improvement Seminar, Washington, D.C., June, 1981.

Podemski, R. "Computer technology and teacher education." Journal for Teacher Education 32(1), 1981.

Podemski, R. "Educational technology and the development-adoption process." Educational Technology 20(5), 1980.

Sheingold, K. Issues related to the implementation of computer technology in schools: a cross-sectional study. Paper presented at NIE Conference in Issues Related to the Implementation of Computer Technology in Schools, February, 1981.

Wolcott, H. "Is there life after technology?" Educational Technology, 21(5), 1981.

ELEMENTARY TEACHER EDUCATION: INCLUDING LOGO IN TEACHING INFORMAL GEOMETRY

by M. Moore and W. Burger

Department of Science and Mathematics Education and Department of Mathematics, Oregon State University, Corvallis, Oregon

Abstract

Elementary education majors have little background knowledge in geometry. Also, there are few opportunities for courses with computer activities integrated in the teaching of the content. Yet, it is recognized that teachers tend to teach using the methods by which they learn. Although students are exposed to one computer literacy course during their senior year, their mathmetics courses do not utilize computer methods. Recognition of these problems directed the revision of the informal geometry course for elementary teachers at Oregon State University. Analysis of Logo exposed strengths and weaknesses in the language for demonstrating geometric concepts. The revised course includes a Logo laboratory in addition to a variety of additional environments for concept development. Evaluation of the course available in June 1983.

In An Agenda for Action: Recommendations for School Mathematics for the 1980s, the National Council of Teachers of Mathematics (NCTM) has addressed the broad issues facing the mathematics education community of the near future. Their first three recommendations are that:

1. problem-solving be the focus of school mathematics in the 1980s;
2. basic skills in mathematics be defined to encompass more than computational facility; and
3. mathematics programs take full advantage of the power of calculators and computers at all grade levels.

Incorporating these objectives into our elementary teacher education program at Oregon State University has been accomplished through a cooperative effort involving the School of Education, the Department of Mathematics, and the Department of Science and Mathematics Education. See (3).

Interpreting the third recommendation as a call for at least minimal computer literacy, a required three-quarter hour Instructional Strategies/Computer Education course was incorporated in the 18-hour mathematics/methods/computer science component. This course has had as its primary goals an introduction to Basic programming and applications of computers in the classroom. As such, the course provides a computer literacy component. The course does provide practical application of strategies in solving problems using the computer. Yet, this course does not intend to teach mathematical concepts other than strategies in problem solving. Furthermore, because of the popularity of the course for all education majors, most students are unable to enroll until their senior year. This aspect unfortunately restricts the possibility of integrating the strategies developed in this course with other mathematics programs for pre-service elementary teachers which would be more in line with recommendation three.

The mathematics content courses have addressed a broad range of topics: problem solving; an informal development of the real numbers and arithmetic; mental, written, and electronic computation; number theory; decimals and percent; ratio and proportion; probability and statistics; and a substantial amount of informal geometry. Topics in informal geometry are included that reflect the spirit of the second NCTM recommendation. These topics enable prospective teachers to develop skills in geometrical awareness and perception, analysis of geometrical shapes, patterns and transformations, and the use of informal deduction (not based on axiomatics) to show relationships among properties of shapes and figures. The specific topics include:

1. types of polygons and their properties
2. abstract measurement
3. geometric constructions
4. patterns of polygons and tessellations
5. motion
6. isometries, congruence and symmetry
7. magnification and similarity
8. polyhedra.

Approximately 13 weeks are spent on these topics.

Our experience has been that many of our elementary education majors have little background knowledge in geometry, and benefit greatly from working in a variety of geometrical environments: graph paper; geoboards, and dot paper; blocks, and cutouts; drawings, and constructions; paper folding; or Logo. This educational method is consistent with learning theory. The abstract notion of symmetry via isometries, for example, is first understood in concrete embodiments, or models, of the concept. Varying the irrelevant attributes, including those that determine the particular environment, and stressing the relevant attributes enables the student to formulate a precise idea of the concept at hand. This

description of learning is advocated by Dienes, Bruner and van Hiele, among others.

Each of the environments mentioned has certain advantages. Graph paper, for example, allows one to study such topics as symmetry in terms of numerical relationships among coordinates of points in the plane. The transformations T_1 $(x,y)=(-x,y)$, T_2 $(x,y)=(x,-y)$, and T_3 $(x,y)=(y,x)$ produce reflectional symmetries, while T_4 $(x,y)=(-x,-y)$ produces another type. Interesting questions arise quite naturally about which types of isometries are "easy" to represent with such transformations. All can be done in the complex plane, but not all are so easy in Cartesian coordinates, the "graph paper" environment. See (2) for other graph paper applications. A possible disadvantage of the graph paper environment is the loss of "hands on" manipulation of objects, and the necessity for a certain amount of prerequisite technical knowledge about coordinate systems, positive and negative numbers, and so on. This serves to point out that each environment facilitates concept formation and the development of problem-solving abilities in certain ways, but that no environment is complete by itself.

Recognition of the strengths offered by using more than one environment led to the interest in including Logo and turtle geometry as one of the environments in the informal geometry course. Additional interest in incorporating the use of the computer in learning mathematics was generated with the recognition that teachers tend to teach using the methods by which they learn. Although students are exposed to a computer literacy program during their senior year, their mathematics courses do not utilize computer methods. The decision was, therefore, to experiment with incorporating turtle geometry and Logo in the informal geometry course because of the special pedagogical advantages of that environment and the usefulness of displaying computer methods of teaching mathematics.

Turtle geometry is a computational style geometry in which the fundamental building block is an entity called a turtle. The turtle, similar to the point in coordinate geometry, is described as having two independent characteristics, a position (as with the point in coordinate geometry) and a heading (which is not a characteristic of the point). The turtle responds to commands which form the language called Turtle Talk. The commands make it possible for turtle transformations in local space rather than in relationship to a fixed global referent. A set of primitive commands provides the language building blocks. All other commands are created using these commands. Therefore, turtle geometry is a mathematical system based on turtle movements. It is a geometry with an undefined term (the turtle), a set of axioms which when implemented in a computer language, Logo, has aspects of coordinate geometry and vector geometry.

Logo is a computer language used for a representation of turtle geometry. The turtle is typically represented by an isosceles triangle. In Logo the turtle becomes what Papert calls "an object to think with." See (8). Use of the computer as a programming tool and the turtle as a geometric drawing device aids the student to combine his own body motions in a familiar space to develop formal geometry.

Adapting ideas expressed by Weir and Watt (9), learning to program in Logo provides an environment for:

1. developing logical thinking and problem solving;
2. using strategies and variables;
3. exploring symmetry, design, angles, geometric forms;
4. developing and testing student's own theories;
5. understanding computers.

An understanding of these characteristics of Logo and turtle geometry encouraged the inclusion of Logo activities in the geometry course for pre-service elementary teachers. The informal geometry course content had previously been carefully defined and described as mathematically necessary in the preparation of elementary teachers. It is important to recognize the intent was not to change the content but to enhance the teaching of the content in another environment.

Thus, it was necessary to evaluate the Logo environment in comparison with other possible environments for each geometric concept. The process exposed strengths and weaknesses in both Logo and other laboratory embodiments. In some cases the capabilities of Logo facilitated concept formation more clearly than other environments. However, in some cases operations in Logo inhibited concentration on the concept.

As an example, consider the construction of the set of regular polygons. In Logo the construction is trivial when compared to the environment offered by use of straightedge and compass. More importantly, the constructions of the regular polygons through Logo procedures emphasizes the properties of the regular polygons in a non-abstract manner.

This Logo procedure (which students create) draws the set of all regular polygons of less than 12 sides (see Figure 1) when called with the command polygon 3.

```
TO POLYGON :N.SIDES
     IF :N.SIDES<3 THEN MAKE "N.SIDES 3
     IF :N.SIDES>12  THEN STOP
     PENUP
     SETXY -100 0
     PENDOWN
     REPEAT :N.SIDES  [FORWARD 50 RIGHT
          (360/:N.SIDES)]
     POLYGON (:N.SIDES +1)
END
```

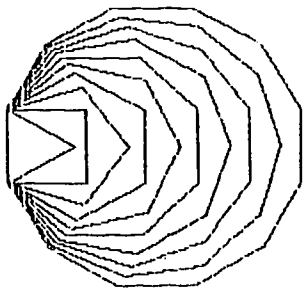


Figure 1

The compass-straightedge environment for the construction of the same ten regular polygons is significantly more abstract although it does present the mathematically interesting Theorem of Gauss. According to this theorem, three polygons in the set of regular polygons with less than 12 sides, the 7-gon, 9-gon, and 11-gon, cannot be constructed using a compass and straightedge. This discovery can provide a nice mathematical generalization but may result in an abstraction for which the students are not prepared when first learning to use the straightedge and compass.

The straightedge and compass procedure for the construction of the regular n-gons begins with the construction of a circle, easily done with the compass. At this point the student must locate the n vertices of the regular n-gon on the circle. A procedure for finding the vertices of the regular pentagon for example, is as follows:

1. Label any point on the circle V_1 and draw $\overline{OB}$ perpendicular to $\overline{OV_1}$.
2. Join V_1 to C, the midpoint of $\overline{OB}$.
3. Bisect angle OCV_1 to obtain the point N, on OV_1.
4. Construct the perpendicular to $\overline{OV_1}$ at N and obtain the point V_2.

The segment V_1 V_2 is one side of a regular pentagon and from it points V_3, V_4, and V_5 can be found. These points are then connected to complete the construction. See (6). Figure 2 demonstrates this construction.

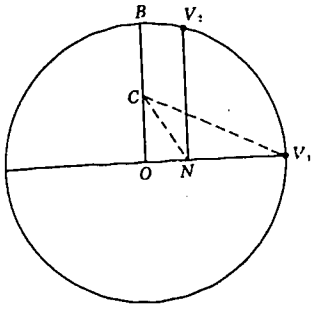


Figure 2

In contrast, the Logo environment can create an inaccurate mathematical understanding when discussing the procedure to construct a circle. Use of the compass to construct a circle embodies the definition of a circle, the set of points equidistant from a specified point. The construction is trivial with a compass. However, in Logo, drawing circles begins with the use of the human body to walk a circle and progress to the procedure, CIRCLE.

```
TO CIRCLE
     REPEAT 360[FORWARD 1 RIGHT 1]
     END
```

Although this procedure "appears" to produce a circle on the output, it is, in fact, a 360-sided regular polygon. The discussion can be extended to include the idea of the circle as the curve approached as the number of sides increases (with the proportional decrease in length). Without some recognition that the "look-alikeness" does not indicate congruence, this example indicates one risk in using only Logo environments for teaching geometry, namely that the student can develop the idea that a 360-sided polygon is a circle.

Another example of the contrast in environments is the problem of triangle construction. Compass and straightedge constructions of triangles allows the discussion of the attributes which completely define a unique triangle. This information is then utilized in showing congruence of triangles (i.e., the three conditions sufficient to determine congruence of two triangles: side-side-side, side-angle-side, and angle-side-angle). Given three sides (meeting the necessary condition that the sum of the lengths of two of the sides is greater than that of the third), construction of the given triangle is trivial with compass and straightedge. Without trigonometry, construction of this triangle in the Logo environment is a significant problem. Elementary teachers typically do not know trigonometry. Therefore, it is not reasonable to change the content of the course to include the discussion needed in order to construct "general", non-equilateral, triangles in Logo. However, the Logo environment provides an excellent environment for the construction of medians, altitudes, and perpendicular bisectors of the sides for any triangle. Halving angles is easily done computationally and then incorporated in a Logo procedure to construct the angle bisectors of a triangle, given the length of each side and the measure of each angle. Similarly procedures can be easily defined for the construction of medians, altitudes and perpendicular bisectors. As a result, the relationshp of the centroid, the orthocenter and the circumcenter can be studied in a less complex manner than through paper folding, protractor and straightedge, or compass and straightedge environments. And, the concentration remains on the concept rather than difficulties presented in the construction.

Logo is particularly well-suited to illustrate concepts in similarity and magnification. Magnification and reduction are easily described in Logo procedures using inputs. For example, this procedure draws squares, (See Figure 3) magnifying them by a factor of two each time.

```
TO SQUARE :SIZE
     IF :SIZE>100 THEN STOP
     IF :SIZE<1 THEN STOP
     REPEAT 4[FORWARD :SIZE RIGHT 90]
     MAKE "SIZE :SIZE*2
     SQUARE :SIZE
END
```

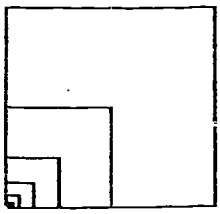


Figure 3

By simply changing the size factor in this procedure to

```
MAKE "SIZE :SIZE*.25
```

the squares will reduce in size by a scale factor of one-fourth.

During spring quarter, 1983, students enrolled in the informal geometry course for elementary teachers will register concurrently for a one credit course, Logo In Informal Geometry. This course will follow the content of the informal geometry course but will provide Logo environments to describe the content. The goals of the course will be as follows:

1. Learning to program in Logo;
2. Having students create Logo procedures which;
   a. construct regular polygons;
   b. construct components of triangles: angle bisectors, medians, perpendicular bisectors of sides, altitudes and the related "centers" of a triangle, incenter, centroid, circumcenter, and orthocenter;
   c. create tessallations of the plane using recursion;
   d. illustrate the isometries of translation and rotation;
   e. construct magnification images of plane figures, both enlargements and reductions using inputs;
   f. combine magnifications with isometries to produce similarity transformations.

The addition of the Logo course will be evaluated in three ways: (1) student success in learning Logo and developing the procedures described above, (2) student success with informal geometry, and (3) student evaluation of the usefulness of the course for prospective elementary teachers. Results of the evaluation will be available from the authors in June 1983.

REFERENCES

1. Agenda for Action: Recommendations for School Mathematics for the 1980s. Reston, Virginia: National Council of Teachers of Mathematics, 1980.
2. Burger, William F. Graph Paper Geometry. Mathematics for the Middle Grades (5-9). Reston, Virginia: National Council of Teachers of Mathematics, 1982.
3. Burger, W. F., Jenkins, L., Moore, M. L., Musser, G., and Smith, K. Teacher Education: A Coordinated Approach. The Arithmetic Teacher, March 1979.
4. van Hiele, P. M. Begrip en Insicht (Understanding and Insight). Dordrecht, The Netuerlands: Muusses Purmerend, 1973.
5. Hoffer, Alan. Geometry is More Than Proof. The Oregon Mathematics Teacher, March 1979.
6. O'Daffer, Phares and Clemens, Stanley. Geometry: An Investigative Approach. Menlo Park, California: Addison Wesley, 1976.
7. Harper, D. O. and Stewart, J. H. (eds). Run: Computer Education. Monterey: Brooks/Cole, 1983.
8. Papert, Seymour. Mindstorms: Children, Computers, and Powerful Ideas. New York:Basic Books, 1980.
9. Steen, L. A. and Albers, D. J. (eds). Teaching Teachers, Teaching Students: Reflections on Mathematical Education. Boston: Birkhauser, 1981.

A COMPUTER LITERACY CURRICULUM FOR PRESERVICE TEACHER EDUCATION CANDIDATES

Brent E. Wholeben, Ph.D., E.M.T., Associate Professor

Department of Educational Administration and Supervision
The University of Texas at El Paso (El Paso, Texas 79968)

With the advent of microcomputers in the classroom to facilitate the ongoing instructional process, an immediate need has simultaneously arisen for the training of the classroom teacher regarding the valid and reliable use of this newest instructional technology. For teacher education candidates who are involved in their semester internship, a unique opportunity exists for providing the necessary preservice computer literacy training coincidentally with their practice teaching experience. As a formal course, computer literacy curricula under the auspices of the College of Education at the university-level must be reviewed for approval by the appropriate state coordinating agency. Such review procedures are often of a year or more in length, yet the need for such training continues to grow. To meet the need for formal instruction in the use of computer technology within the classroom, alternate methods must be developed for beginning teacher training.

The Emergent Learning Technologist

With the advent of the microcomputer for computer-assisted instruction (CAI), the beginning classroom teacher has available the most formidable ally for the conduct of teaching since the printing press. The computer has been shown to be capable of selectively facilitating the instructional efforts of the teacher in such a way as to provide greatly enhanced learning on the part of the student. But the use of CAI in producing increased learning can only be made possible through the careful training of the teacher-user.

The teacher today must therefore be viewed as a learning technologist -- a professional whose role must assume the structured intervention of computerassisted technology within the learning process of the child. To accomplish a successful intervention however, a careful process of training must ensue for instructing the teacher how to best utilize CAI, for whom, when, and why. To accomplish these several missions, the needs of the teacher and student alike must be addressed and understood.

The beginning classroom teacher must have a basic understanding (or general literacy knowledge) of the microcomputer, and its application to the processes of the instructional classroom. In addition, this same teacher must have the ability to apply the use of the computer for both CAI as well as the management of that instruction (CMI). As school districts continue to increase their expenditures for the purchase of microcomputers for classroom use, teachers must fully understand the potential application of this new instructional technology.

The student has basic needs also related to the employment of computerized instructional technology. Although many needs address the issues surrounding increased effectiveness, efficiency and satisfaction regarding study and learning of appropriate curricula, other needs require attention apart from the school environment itself. Students exist in a technologically-oriented world, and therefore must be helped to cope adequately with such demands as will be placed upon them by a technological society. Such objectives as understanding the potential of computers (both positive and negative), how computerized technology is currently utilized to benefit mankind, and where computers could be utilized to invade privacy rights of the individual -- can only come from understanding the computer itself.

Training at the Undergraduate Level

It is the mission of this paper, to examine the potential for training the beginning classroom teacher during the classroom internship cycle at the undergraduate teacher education level of university training. While the role of an inservice paradigm will always be necessary for continued on-the-job training of the classroom teacher, the cost-effective rationale for including computer and CAI/CMI literacy training at the preservice teacher education level exists in three areas.

First, preservice training at the undergraduate, teacher education level provides systematic quality control related to the information presented regarding instructional technology literacy. Secondly, the opportunity for providing sufficient time for the structured implementation of required learning activities exists during the semester of the student-teacher internship. Lastly, this same time will provide the necessary opportunities for the preservice teacher to prac-

tice various CAI approaches while in a regular classroom (under the supervision of the regular teacher) and to report on the varying success experienced -- promoting optimal information

transfer to the student teacher.

Rationale for a Variable Instructional Sequence

As will be presented in the next section, the proposed presentation of the literacy objectives and activities occurs in a variable time sequence; that is, activities and meeting times will vary depending upon the material to be presented, and when the actual presentation will take place. Four issues of rationale are apparent in requiring such a variable instructional format.

First, the literacy (instructional) objectives and activities are themselves of variable-time format, requiring significantly different durations to effectively present the material. Secondly, these same objectives and activities are linearly sequential. Therefore, rearrangement of activity sequence is precluded. Thirdly, the successful assimilation of the material requires appropriate time lapses between activity presentation(s). Lastly, these same time lapses permit ample opportunity for student teacher practice and curricular development intervention(s) during their classroom teaching activities.

Components of the Semester Curriculum

The implementation of the computer literacy curricular program for teacher education candidates will exist over a single semester, preferably the semester within which the candidate is performing the required internship (practice teaching) experience.

The components of the semester-length program exist in three parts, each representing a serial stage in the development of awareness, skills and personal techniques concerning the utilization of computers for instruction. These three parts are defined as:

1.0 The Awareness Component (a single session, four-hour sequence of activities designed to introduce the participant to the idea of microcomputing, and its potential for classroom instruction utilization; and to provide a structured, first-stage hands-on experience for each participant);

2.0 The Developmental Skills Component (a four-session sequence of two-hours each, during which the participant is provided an in-depth coverage of those topics initially introduced within the preceding 'awareness' session, including such topics as: CAI/CMI intervention techniques, procedures in the assessment and selection of instructional software and hardware, demonstration of various curricular oriented CAI packages from available vendors, and responsibilities associated with the employment of instructional technology within the classroom and learning process); and,

3.0 The CAI Incorporation Component (a three-day professional workshop of 12-sessions of 1½-hours each, during which participants are afforded opportunities to explore the use of CAI/CMI software in their classrooms as first-year teachers (the following semester or year), to learn techniques in the design of specialized lesson plans for CAI applications, to understand the basic rudiments of computer programming for specialized applications in the classroom, to interact with colleagues of like grade-level and discipline areas concerning the incorporation of CAI within the classroom process, and to research with local vendors the potential of microcomputers for facilitating the instructional process.

Component A: Awareness

The first component in the series of computer literacy instruction for the preservice teacher education candidate is intended as a first-stage introduction to microcomputers and their potential for use in the classroom, thereby providing the initial foundation for awareness development on the part of the future beginning teacher. A sample scheduling of activities for this four-hour block of instruction exists as follows:

Activity 1	(0:30)	Introduction to Microcomputers, and Their Potential Utilization within the Schools;
Activity 2	(0:45)	Introduction to the Components of Microcomputer Hardware and Software, and Their Terminology;
Activity 3	(0:30)	< audio-visual presentation >
	(0:15)	** break **
Activity 4	(1:30)	Demonstration(s) of Appropriate Microcomputer Hardware and Software (round-robin format of 15-minutes per each station), and including such topics as: programming, simulation, CAI packages, CMI programs, and color graphics capabilities;
Activity 5	(0:30)	Question/Answer Period (small or large group), and Post-Session(s) Evaluation and Feedback.

Component B: Skill Development

This second component in the series of computer literacy topics for preservice teacher education candidates exists as a direct extension of those topics covered during the first (awareness) component.

During the skill development component, four individual sessions of two-hours each are employed to provide greater in-depth treatment of those topics introduced previously. Conducted as single evening sessions in a seminar format over the course of four-weeks, sample topics for these four-on-two survey sessions will include:

1.0 INTRODUCTION TO MICROCOMPUTER APPLICATIONS FOR THE CLASSROOM

- 1.1 Data-Processing via the Microcomputer
- 1.2 Computing Needs Assessment for Classroom Activities

2.0 INTRODUCTION TO MICROCOMPUTER HARDWARE DESIGNS AND CONFIGURATIONS

- 2.1 Components of Microcomputer Hardware Design
- 2.2 Operation of the Microcomputer Hardware System

3.0 INTRODUCTION TO MICROCOMPUTER SOFTWARE APPLICATIONS FOR TEACHING

- 3.1 Components of Microcomputer Software for Classroom Use
- 3.2 Demonstration of Instructional Software Applications

4.0 INTRODUCTION TO SCHOOL MICROCOMPUTER SELECTION AND ACQUISITION

- 4.1 Preparation of the Decision Matrix for Microcomputer Hardware and Software Selection
- 4.2 Solicitation and Validation Strategies for Area Vendors

Component C: CAI Incorporation

The third and final component of the computer literacy program for preservice teacher education candidates involves a three-day sequence of 12 sessions of 1½-hours each. This phase of computer literacy training is designed to acquaint the future beginning teacher with the skills and procedures necessary to actually utilize computer-assisted instruction within their particular discipline at the classroom level.

The incorporation stage of computer literacy training commences with a full day of intensive lectures on the use and design of CAI/CMI-oriented instruction. In addition, two sessions allow suitable opportunities for students to acquaint themselves with instructional software specific to their teaching curriculum. Group activities for the sharing of information (reaction to the software, suitability of the packages for instruction at the particular grade level, and the potential for CAI intervention in specific instances) are structured at the end of each demonstration sequence.

The second day is concerned mainly with the specifics of planning, designing and implementing CAI strategies for classroom application. These sessions include such topics as: how to delineate curriculum for CAI-intervention structuring, how to develop the CAI-oriented lesson plan, and how to apply CAI to special populations and/or interest groups.

The third day primarily focuses upon the content and process associated with the evaluation of microcomputer software and hardware for matching the intended goals of instructional use. Special interest groups allow the further sharing of specialized information, and provides a basis for evaluating the perceived benefits of the three-day sequence. Finally, vendor demonstrations and presentations at the conclusion of the incorporation phase provide the students with the basis for designed their equipment need requests for the upcoming school year.

A sample outline of this three-day incorporation component follows:

Day-1/Ses-1 Presentation:

Design of CAI-Oriented Instruction as a Classroom Teaching Strategy;

Suggested Presentation Topics:
A. six-stages of computer-assisted instructional intervention
B. three-levels of computer-managed instructional monitoring
C. cross-reference matrix for CAI and CMI development

Ses-2 Individual Simulations:

Students interact with suitable software packages; and participate in small-group, follow-up discussion;

Ses-3 Presentation:

Design of CAI Programs -- What to Expect and What to Require;

Suggested Presentation Topics:
A. student-user orientation
B. menu-driven routines
C. compatibility of instructional objectives
D. testing and progress monitoring

Ses-4 Individual Simulations:

Students continue their exploration of suitable software packages related to their specialized needs; and participate in small-group discussion;

Day-2/Ses-1 Presentation:

Techniques in Applying CAI Packages for Instructional Facilitation;

Suggested Presentation Topics:
A. individual student versus small-group orientation
B. tutorial versus remedial versus enrichment modes

Ses-2 Presentation:

How to Delineate Curriculum for Identifying CAI-Oriented Potential;

Suggested Presentation Topics:
A. clarification of needs versus desires
B. establishment of concept mission, instructional goals, classroom activities, and student tasks

Practicum Laboratory:

Students are afforded the immediate opportunity to simulate curricular delineation relative to their specialized areas;

Ses-3 Individual Simulations:

Students continue their exploration of instructional software, researching software suitability based upon their recently delineated curricular objectives;

Ses-4 Presentation:

How to Design and Implement the CAI-Applications Lesson Plan;

Suggested Presentation Topics:
A. considerations of equipment and material availability
B. considerations of time, facilities, and varying student needs

Practicum Laboratory:

Students are afforded the opportunity to design a simulated lesson plan based upon their knowledge of CAI programs previously explored during the individual simulation sessions;

Day-3/Ses-1 Presentation:

Special Techniques for the Evaluation of Instructional Software and Compatible Machine Hardware;

Suggested Presentation Topics:
A. content and process criterion references for software quality
B. process and tooling criterion references for hardware quality
C. four steps in approaching the software/hardware compatibility assessment

Ses-2 Special Interest Group Sessions:

Students meet by grade-level and curricular discipline areas to discuss common interests and needs related to CAI application; and a formal evaluation of the three-day sequence is conducted;

Ses-3 Vendor Demonstration of Software:

(round-robin scheduling);

Ses-4 Vendor Demonstration of Hardware:

(round-robin scheduling).

Scheduling of the Semester Program

Although differences exist in terms of time-availability at various institutions regarding their instructional sequence (semester, quarter, trimester), little modification to the computer literacy program presented would be required.

It is suggested however, that the program be implemented as early as possible during the 'internship semester' -- preferably prior to the mid-term period. Such early intervention provides maximum opportunities for the student-teacher to employ the full range of skills learned during the preservice training sessions.

It is also suggested that skill laboratories be scheduled during the second part of the 'semester period'. Such sessions will provide opportunities for preservice teachers to address and rectify problems which have developed during their initial CAI incorporation efforts within the classroom.

DYNAMICS OF LEARNING AND MISLEARNING IN A SIMULATED MICRO-WORLD

Andrea L. Petitto & James A. Levin

Graduate School of Education and Human Developement
University of Rochester
Rochester, New York 14627

Center for Human Information Processing
Unvisity of California, San Diego
La Jolla, CA 92093

Abstract

This paper presents an overview of research on children's learning processes in a computer implemented micro-environment. A class of fourth and fifth grade children played a set of "shark shooting" games as part of their regular school activities for a three month period. The games required the estimation of numerical values on number lines, and the coordination of vertical and horizontal dimensions. Observations are made concerning variations in the development of game skills, and transfer of number concepts to non-computer activities. Discussion focuses on transcript analyses revealing specifics of learning processes during play. Interactions among cognitive skills, game features, and the role of goals in structuring conceptual development are discussed.

Introduction

At the Laboratory for Comparative Human Cognition at the University of California in San Diego, several of us have been investigating cognitive implications of the use of micro-computers in elementary classrooms. Some of this work has concentrated on the use of computers as learning environments. In order to understand how learning occurs in these environments, we have found it necessary to look closely at the details of the interactions among the children, the various helpful adults in the room, and the computer itself. In this paper we describe a study of learning in a simulated micro-environment in which we take a close look at the qualitative differences in performance between the most and least successful players.

A well known example of a simulated micro-environment is diSessa's dynaturtle which behaves strictly according to Newton's laws of motion.

This research was supported by the National Science Foundation, Research in Science Education Grant SED-8112645. Many thanks go to Robert Rowe, Marcia Boruta, Karen Johnson, Jose Vasconcellos and Dan Rieswig for their help and support.

The main idea behind the use of such simulations is that they embody general principles and relationships in the material to be learned and permit a kind of exploratory behavior that is not usually possible when the same material is presented in standard, expository written or spoken formats. Students explored the properties of this Newtonian object by manipulating it and developing strategies to manage it. Exploration is not simply a matter of trial and error, as might be encountered in drill and practice. In a computer implemented simulation feedback from an error or an exploratory attempt is informative. That is, it not only provides information about accuracy, but gives additional information about relationships embedded in the micro-environment.

The type of simulation we will be concerned with here takes the form of an educational game. Games such as these usually simulate micro-environments which incorporate a goal. In the dynaturtle game, for example, the Newtonian turtle is to be guided to a specific "port", or in another version, around a circular track without crashing. The point of introducing game versions of simulated environments is that they require players to develop and sharpen cognitive skills in the service of attaining the game goal. Because of the inclusion of goals internal to the system, games are more able to stand alone than are pure simulations which students manipulate to achieve externally derived academic goals. The inclusion of internally defined goals also affords the opportunity for an internal tutor function. Since the program can assume that the goal of the player is the internally defined goal of the game, it can monitor the effectiveness of the player's performance. This allows the program to recognize and respond to errors by offering hints, and can alter game parameters to adjust automatically to different levels of skill.

The Shark Games

Working on the notion of exploratory activity, several of us have developed a family of three estimation games called the "Shark Games". The games are intended to strengthen children's knowledge of numerical relationships. Earlier work by us and by other researchers had found that fundamental concepts of numerical rela-

tionships are often weak among poor achievers in arithmeticl. We hoped that these games could strengthen these fundamental concepts directly, and that this would in turn affect classroom arithmetic skills.

All the Shark Games utilize the same game world - sailing on the high seas, hunting down and harpooning sharks. All three games require estimation along two dimensions marked by numerical scales (see Figure 1). All three of the games were used in our research, though for the sake of brevity only one, called "Sonar", will be described here. In Sonar, the player is sees a pair of coordinate lines - one horizontal and the other vertical - on the screen display. The location of an invisible (underwater) shark is indicated by a pair of numerical coordinates written at the top of the screen. Using these coordinates, the player must estimate the shark's position visually. Each dimension is dealt with in turn, first the horizontal estimate (labeled "aim") then after the aim is set the vertical dimension (labeled "distance") is entered. In effect, each of the coordinate lines serves a dual purpose, as an axis on one dimension and as an indicator marking a numerical selection on the other dimension. The player uses game-paddles (though keyboard entry is a possible option) to move indicator lines indicating the position he thinks is specified by the numerical information. The paddle button (or RETURN) is pressed to "set" each numerical estimate once it is made. To avoid the confusion that can arise from the dual role of each axis line, while each dimension is being estimated, the endpoint numbers and label on the other axis line (now acting as an indicator) disappear.

The "throw" of the Harpoon, "hits" and "misses" all are represented in an interesting graphic display with sound accompaniment. Once both dimensions are entered, a "harpoon" moves from the bottom of the screen to the point specified by the intersection of the horizontal and vertical estimates. Feedback is in the form of a "splash", visually specifying the actual location of each throw. The splash is labeled with its actual numerical coordinates and remains visible on the screen as an additional point of reference through the next several tries.

The games also perform a tutor function. When a player's shot misses the shark, verbal and directional hints are written at the top of the screen. When a player's estimate is too high, the word "smaller" with an arrow pointing in the direction of lower numbers (<=) for that dimension. If the estimate is too low, the screen displayes "bigger" with an arrow pointing in the direction of higher numbers (=>) for that dimension.

To adjust for variations in skill, the games all included multiple levels of difficulty such that the computer took over some functions at the easier levels. This was accomplished by creating "beginner" levels in which only one axis (horizontal or vertical) is used, while higher levels involve two coordinates. Difficulty on the two axis games is varied by reducing the size of the shark with respect to the numerical span of the axis, thus requiring progressively more accurate "thows". Movement to higher or lower levels on successive games is automatic, contingent upon rate of success.

The Study

Our original purpose in carrying out the study was to examine the possibilty of transfer of game-related skills in numerical estimation to other kinds of numerical manipulations, particularly paper and pencil tests of number line skills and classroom arithmetic. It is impossible to overlook the variation in academic achievement that is typically found in most elementary school classrooms. For this reasons, we were also interested in finding out how learning to play the simulation-games themselves interacts with these differences in academic skill. Thus we were looking for a two-way effect, game skills transfering to classroom performance, and academic skills affecting the ability to become skillful in the game.

Two structural aspects of these games were particularly interesting to us from a theoretical perspective. One is the inherent requirement for successive approximations2, and the other is the introduction of cartesian coordinates requiring the coordination of two dimensions. Strategies which use successive approximation are encouraged by the game setup which allows for multiple tries (or throws) with informative feedback accompanying each miss. The coordination of estimates with respect to the two axis system was accomplished through a social coordination between two players. Children usually played these games in pairs, one child of each pair playing one of the dimensions. We were interested in the way that the social coordination of action might serve as a basis for spatial coordination on a two-dimensional plane.

With all this in mind, we placed the Shark games in a combined fourth-fifth grade classroom where they were used in a computer-based "center" scheduled to be visited by specific pairs of children each day. In this way, all the children in the class were able to work with the Shark games on a regular basis, amounting to about one half-hour per child per week over a period of three months. The Sonar game was first introduced into the classroom with a range of 0 to 100 on each axis.

At the first session, and at several important transitions throughout the study, an observer/helper attended the children's shark game sessions. As the term implies, the helper/observer had a dual role. One role was to record events as they occured during game play. The other role was to help the children if they encountered any major difficulties preventing them from effectively playing the games. The specifications of the helping role were to keep the game going while interfering as little as

possible. A graded sequence of hinting procedures was to be used when intervention was necessary. First, the observer/helper was to simply point out relevant information on the monitor screen. Then, if that were not enough, point out relevant relationships between game elements. And finally, if all else failed make suggestions about what to do.

Several kinds of data were collected on game play itself. The games had been set up to record all sequences of keypresses, all game parameter values, and time information from an internal clock. This data was taken on all games played throughout the study. There were also the audio-tapes and field notes taken at several transitional points during the study. The audio-tapes recorded conversation and other sounds and could later be coordinated with the written field notes and computer collected records.

Results

General Trends

Most children increased their skill at playing the shark games themselves throughout this time period, though there was considerable variability among children in how well they played. Our findings showed the classic pattern - those who played relatively well in the beginning also improved more than those who played more poorly on the first few attempts. Nevertheless, even with the skilled players, we could find no overall transfer of game skill to paper and pencil tests which assessed number line and written arithmetic skills.

These findings were quite disappointing since we had developed these games specifically to help the poorer students, and had hoped that game derived skills would show transfer to some paper and pencil tests. We decided to look at the children's game playing in some detail to find out what had actually happened.

A preliminary analysis has been done on the keypress data for the initial session and on overall achievement within the game itself throughout the study. Overall game achievement was assessed as the percent of sessions in which the players attained level 6 or better (out of a possible 9 levels). This method of scoring overall game achievement resulted in scores for 19 individual children ranging from 10 to 90%. We then designated the 7 players who scored over 70% as "high", the 6 players with scores between 40 and 69 as "intermediate", and the 6 below 30% as "low" in overall game achievement. (There were no scores between 30 and 40 percent.) This presentation concentrates on the contrast between the high and low game players. We found that in the introductory session low achievers overall had spent more time on the single-dimension lower levels; used more "throws" per game to hit the shark; and showed a much higher average deviation of throw values within each game. In general, the overall poorer game players appeared to be less systematic in their initial approach to the games.

Transcript Analysis

In order to account for these data, we looked at the interactions among the players, the observer/helper, and computer as recorded by our audio-tapes and field notes. We found that there were at least two major ways that game strategies differed between players ranked as good or poor in overall game achievement: the emphasis on numerical judgements in making game decisions, and the ability to learn and manage the mechanics of the game. Differences in the use of numerical strategies were striking. The better players used numerical specifications frequently as a way of explaining thier own actions or specifying some game related information to another player. But numerical judgements did not substantially enter into the remarks or game decisions of the poorer players.

It is not surprising that players with poorer number concepts should use numerical information less than those with better understanding of numerical relationships. But the mechanics of the shark games should be just as unfamiliar to all the children, regardless of numerical skills. Nevertheless, the poorer players also appeared to have more difficulty adjusting to the mechanics of playing the game itself - how to set up and execute the shots, coordinate the actions of both players on the two-dimensional levels, and so on. Could the lack of certain conceptual knowledge interfere with the ability to learn the mechanics of a game in which that knowledge must be employed? Or is there some other reason, perhaps some fundamental problem underlying both poor mathematical ability and game learning ability?

The data from this one study can not provide definitive answers to these questions. But an analysis of the details of the children's actions in the games reveals some interesting relationships among cognitive skills, understanding of specific relationships among game elements, and recognizing the goal of the game.

All the children in this study quite easily recognized the main goal of the game - to shoot a shark with the harpoon. There appeared to be no difficulty accepting the moving arrow as a harpoon or the idea of a hidden shark which, when hit, appears momentarily as a small triangular dorsal fin. The unanimous acceptance of this game goal, however, masks a host of ambiguities which only become apparent from the children's remarks and questions addressed to each other and to the observer/helper. Because the misconceptions arising from these ambiguities prove to be functionally related to the development of cognitive skills, two representative examples of these misconceptions are discussed here at some length.

Some Failures: Several children showed considerable difficulty even on the low level single-dimension games. After completing seven

single-dimension games and in the middle of multiple attempts to hit the shark during the eighth game, one such child asked if the shark were "roaming around in there". In this context, the remark suggested that the player thought the shark was a moving target, invisibly swimming across the screen. Such a notion could derive from repeated frustrated attempts to estimate the shark's position, or it might be a misconception from the start. After all, sharks are not characteristically stationary, especially when under fire. However it arises, the idea of a moving target calls into question the player's understanding of the relevance of the numerical specification of the shark's position. If the shark is moving, then the numerical information is no longer relevant, and Sonar becomes a guessing game in which the "smaller", "larger" hints provide the only clues. This pattern essentially characterized the game playing of several of the poorer players. Instead of using numerical information to progressively narrow down the search for the shark (the process of successive approximations we were interested in), these players simply used the directional information from the hints without calibrating the distances or coordinating successive tries in any way at all.

Notice that this erroneous conceptualization makes fundamental changes in the underlying goal of the game. Specifically, it makes irrelevant the major goal that the game is intended to support: get as close as possible to the position on the screen which corresponds to the numerical coordinates printed in the upper part of the display. When a player does not recognize the goal of a game, no amount of practice will bring him closer to achieving it. The child has missed the point and is simply playing a different game, one involving different skills, strategies and goals than those intended by the game's designer.

There were many other technical pitfalls that lead to major misconceptions about the game and its goals. One which proved to be a destructive factor as the game progressed concerned the function of the RETURN key. Programmers and users of software tend to have differing views of the function of the RETURN (or ENTER) key. From a user's point of view, the RETURN key appears to initiate an action, to start something. Where the programmer intends the RETURN keypress to tell the program to read a line of characters just typed in, the user sees it as a keypress that directly starts an action. The tacit assumption on the part of a naive user is that the computer is reading along as information is entered at the keyboard or other peripheral input device. The RETURN key simply starts the next machine function.

In the Shark games, the introductory text and instructions end with the instruction: "Push RETURN to start a game", further reinforcing the idea of RETURN as "begin". Not surprisingly, more than one of the children in this study treated the RETURN key as an initiator. In at least on case, this became a rather pernicious bug in the child-computer interactions. The problem first showed up when one child in a pair of players read aloud the screen instruction: "Set the aim, then push RETURN." The second child then repeated the instruction in the following form: "Then push RETURN? Push RETURN? Then I start the aim?" Though the first child and the observer/helper manipulated the situation so that the aim was at least sometimes set before RETURN was pressed, the "RETURN starts things" idea persisted into the higher, two-dimension levels of the game. This player would then press RETURN to "start" her partner's turn, thereby firing the harpoon before he could make any move to set his own aim. This short-circuited the possibility of cooperative, coordinated activity and the children in this case (as in at least one other) persisted in thinking they were playing competitively against each other. The goal of coordinating two dimensions was never established.

These examples are typical of the difficulties the poorer players had learning the mechanics of the Shark games. In some cases, specific misconceptions were quickly corrected by adult intervention. In other cases, misconceptions persisted through several sessions and some were never resolved. These technical difficulties, though they are not directly related to the concepts we wanted the children to learn, reorganized the dynamics of the games so that they no longer addressed the relationships the programs were intended to embody.

This line of reasoning suggests that technical misconceptions on the part of some players reduced the effectiveness of these games in promoting the development of numerical concepts. But evidence from the better players indicates that the causality implicit in the above statement might also run the other way. Weak numerical concepts might have left the poorer players open to technical misconceptions by not providing a coherent framework to guide learning of the game.

Some Successes: In the introductory sessions, as in subsequent ones, the better players overtly and verbally refered to numerical information in all aspects of play: in aiming the harpoon themselves or in guiding each other's actions, "Seventy-eight is about here"; in taking roles in two- dimensional play "You got sixty-four", and so on. This often lead to successive approximation strategies. After two misses, for example, one player remarked, refering explicitly to the value of the second miss and tacitly to the first: "Twenty-one. We have to be right between here."

The numerically specified position information and the familiar (though tacit) numerical relationships on the number line provided a coherent structure within which the technical aspects of the game could be worked out. That is, possible ambiguities such as the function of the RETURN key and the nature of the shark's

movements are worked out in the service of maintaining coherence within the numerical context. The process of working out the technical details of the games usually went so smoothly among the better players that it was often difficult to detect transitions to serve as examples. A few observations can be cited here to illustrate the point.

In one case, a player had begun by randomly aiming and shooting across the screen, apparently without regard for numerical information. After two low-level games in this mode, the observer/helper simply pointed out that the shark's position was numerically specified. The player's next aim was again wild, but this time she verbally predicted it's numerical value, saying "That's probably gonna be two." This prediction signaled her first attempt to coordinate position and numerical information, and her accuracy increased steadily with subsequent tries.

In another case, we discovered that a player had not clearly understood that the hidden shark was not a moving but a stable target until his ninth game. Though he had participated effectively in quite accurate and strategic play in all these games, this boy had been setting his aim according to the specified numerical informaton, not recognizing its relationship to the position of the hidden shark. During the ninth game, he expressed surprise that the shark so often turned up where he had shot: "So wherever you hit the thing, it goes?" This player had also assumed that the shark was swimming invisibly around the screen, but in contrast to the previous case where the notion of a moving shark was so debilitating, this boy's playing had been consistently strategic and effective. This was possible because he was using numerical information to structure his activity before he completely understood how all the game elements were interrelated. Ultimately, he discovered the "stable shark" feature because his actions were consistently guided by appropriate goals which were supported by his understanding of the very numerical concepts that the game was designed to teach.

Discussion

Though this very preliminary analysis of our recently gathered data cannot give definitive evidence about learning processes in interactive media, several important points can be made. First, stand-alone simulation-games cannot be considered a panacea for the problem of remediation of low achieving children. The specific game-goal of shooting sharks was not by itself enough to organize behavior in ways that would lead to development of the intended skill. In order to insure that the players interpreted the game the way the game designers had intended, some adult intervention was necessary even for the best players. Initial misconceptions are difficult to alter once they are established3, and can be perniciously detrimental.

It can be argued that these difficulties arose because of flaws in software design or that better tutor functions would more effectively monitor and guide a player's actions. Both of these arguments are probably valid. However, we cannot assume that software - however well designed - will ever be completely free of either of these problems.

When unfamiliar goals are introduced requiring a new application of undeveloped or underdeveloped skills, it can not be assumed that a novice player will apprehend all the relevant relationships among game elements and correctly infer the goal or goals intended by the game's designers, however clearly these goals appear to be presented. If a player infers a wrong goal, then feedback from an automatic tutor is not likely to be effective. This feedback is only corrective with respect to those goals the game designers intended. But players would interpret such feedback in terms of their own intentions. If a goal is misconstrued, then automatic tutor functions are rarely adequate to correct it. Continued practice in the game rarely corrects this misunderstanding and usually causes it to be more deeply entrenched. At the same time, since changes in game goals alter the dynamics of the game itself, the skills and strategies that are being developed through practice may not be those which the designers of the game or the teacher intended.

These principles might at least partly explain the lack of transfer from game playing to paper and pencil analogs of number-line estimation tasks. With inadequate supervision at entry into the game environment, children whose number line skills were too poor to provide a way of recognizing the salient structures of the game did not play the game as it was intended to be played. They effectively rearranged the dynamics of it to avoid practicing those skills we wanted them to develop. Children whose number concepts were sufficiently strong to recognize the salient relationship expressed numerically generally did learn the game as it was intended, but did not need to improve those skills very much to play. At least not enough to be detected by pre- and post-tests.

Conclusions

The function of education is to transfer to children socially organized and formalized knowledge - in the case of this study, the number system and the arithmetic and coordinate systems based on it. Thus far, our analysis is able to show that when these knowledge systems form the basis for a simulation or simulation-game, players must rely either upon prior familiarity with that system, or external guidance to discover the relevant parameters for manipulation. We expect to show from further analysis and in future research efforts that once this basic understanding is established exploratory activity can be productive.

Beyond this, our observations have implica-

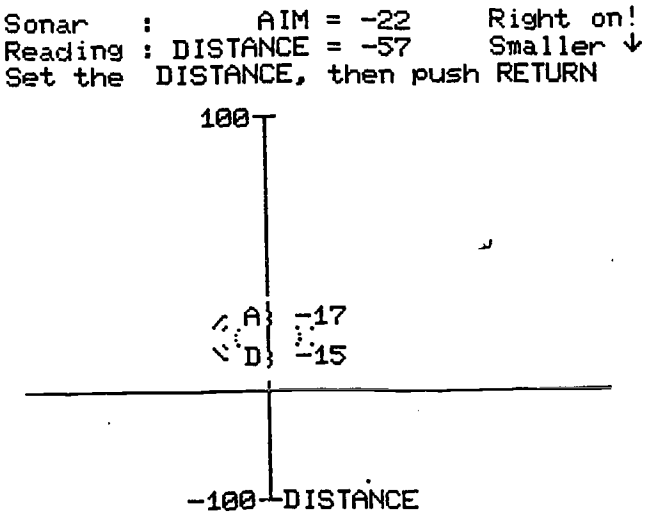
tions for educational practice. They suggest that it is important to consider the role of the teacher when investigating the dynamics of child-computer interactions. The use of computer-based media might provide a new role for teachers. With computers, parameters and rules for manipulation are internal to the machine. They are not unambiguous, and novices - either children or older students - need help to learn to manipulate them effectively. Because of this, the teacher can become an ally and helper to the student, a role which contrasts to the usual one of task master and judge. Future research should investigate the potential of this teacher-student-computer interaction system in which the teacher's competence serves as a resource for students in problem solving situations.

References

1. Petitto, A. L. "Developmental study of arithmetic competence among children with school related learning difficulties", unpublished working paper, Laboratory of Comparative Human Cognition, Univeristy of California, San Diego, 1982.

2. Petitto, A. L. "Long division of labor: in support of an interactive learning theory", unpublished manuscript, Graduate School of Education and Human Development, University of Rochester, Rochester, NY, 1983.

3. Miyake, N. Constructive Interaction, CHIP report #113, ONR report #8206, Center for Human Information Processing, University of California, San Diego, 1982.



igur.

This is a typical screen display in Sonar. The numbers at the center top of the screen indicate that the shark is hidden at -22 on the AIM (horizontal) line and -57 on the DISTANCE (vertical) line. The player has already made one shot which has left a "splash" (ragged concentric ovals) at -17 on the AIM dimension and -15 on DISTANCE. Verbal hints appear to the right side of the top of the screen. "Right on!" indicates that the AIM estimate (-17 showing in the "splash") was close enough. "Smaller" indicates that the DISTANCE estimate (-15 showing in the splash) needs to be revised downward. The downward pointing arrow indicates the direction that the indicator line must be moved.

Note that the AIM line is not labeled here. This is because the player is in the process of selecting an estimate on the DISTANCE dimension and the AIM line is for the moment acting as an indicator rather than a numberline.

Observation and Inference - A Computer Based Learning Module

by Alfred Bork and David Trowbridge
Educational Technology Center, University of California, Irvine

Arnold Arons
Department of Physics, University of Washington

Abstract

This paper reviews a computer dialog to teach the distinction between observation and inference. It is a self-contained program designed to work with a wide range of students.

An important distinction in undertaking the nature of scientific knowledge is that between what is observed, seen directly, and that which is inferred from the observed evidence. While this distinction is, like all human distinctions, not absolute, it is nevertheless extremely useful in understanding the nature of scientific information. It might also be considered an important intellectual tool, one we want to bring to students at as young an age as possible in order to enhance intellectual development.

Experience shows, however, that many students at all levels, have difficulties making the distinction between what is seen, and what is reasoned. Even at the college level many students do poorly on examples of this type. Some of the creationist literature furnishes striking examples of fuzzy distinctions, exhibiting a failure to distinguish observation from inference.

The same kind of distinction arises in other disciplines. In studying history, for example, it is necessary to distinguish between primary information or evidence on the one hand and interpretations or inferences drawn from such material by the historian on the other.

The program to be described here is a computer based learning module, intended for students from about 12 yeaers of age or over, concerning the distinction between observation and inference. It involves a variety of situations to illustrate and establish the distinction. This project was funded by the National Science Foundation, through the Development in Science Education Program. It is primarily concerned with aiding students in early adolescence, about the age of 12, to develop intellectual skills which are important for later life. Although initially the primary focus was on various standard Piagetian tasks, we have also considered other exercises in abstract logical reasoning such as the one presented in the present paper. This program has been tested with a very wide range of students, in several different environments, as will be discussed.

Environment

This module is about 15 to 20 minutes long for the average student. It is embedded in a longer program called "Spacelab." The longer program presents a fantasy about space travel in which the student is the captain of a starship which comes across another, derelict, starship carrying some supposed "energy crystals." Measurements of mass, volume and other properties are made on these energy crystals to try to decide which ones might be a possible source of energy (Fig. 1). Students are introduced to the concept of density and use this idea to search for crystals consisting of the same material. From the standpoint of Piagetian tasks, the

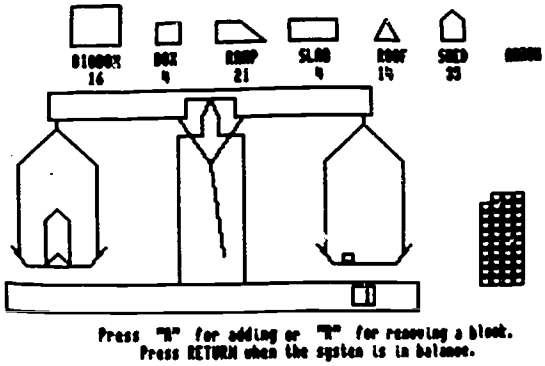


Figure 1

A weighing experiment in the Spacelab dialog.

exercise is principally concerned with ratio reasoning, involving the ratio mass/volume, but an opportunity arises to lead the student into making the distinction between what is observed and what is reasoned.

The observation-inference module comes as

the last sequence in "Spacelab," but it can also be used independently of the larger program in which it is embedded. In fact, we have two versions of the module, one in which there are references to the "parent" Spacelab program and one in which there are no such references, so that the module can be used alone.

Program Outline

The program begins with a sequence from a Sherlock Holmes novel, "The Greek Interpreter," by Arthur Conan Doyle. There are many passages in the Sherlock Holmes novel in which Holmes, or in this case, his brother Mycroft, make a series of startling deductions about a person or situation based on what appear to be relatively few, apparently insignificant, observations. We have picked one of these passages to begin the current program (Fig. 2). The entire passage is first presented to the student in an attractive

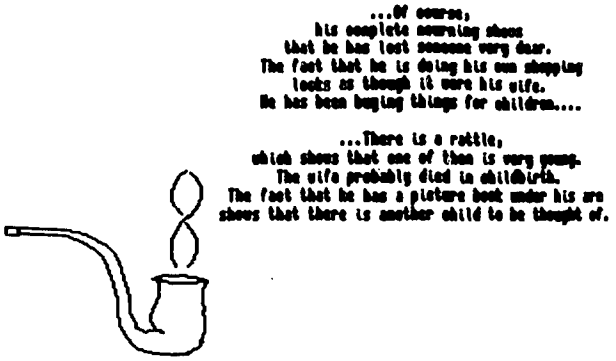


Figure 2

Excerpt from a Sherlock Holmes story.

way and with some associated visual information. Then the program analyzes the passage, classifying for the student several examples of what is seen directly and what is reasoned out. In each case, this is done by using a blinking box surrounding a phrase or sentence of text, and then classifying what is in the blinking box. At this point the computer is being used in an expository manner, but this is only a short episode. Note that we are not using the "technical" terms, observation and inference, as yet.

The second activity in the Sherlock Holmes sequence requires the student to make his own classification of items still remaining in the passage. The blinking box is still used to set off a phrase or sentence, but now the student must decide, on the basis of the earlier example, whether each item refers to something that Mycroft sees or to something which Mycroft reasons out. If mistakes are made, they are corrected.

When the examples in the passage are exhausted, we finally introduce the words "observation and inference" (Fig. 3). In much of our materials at Irvine we have taken taken care to avoid using a technical term until the idea it denotes has been established operationally through specific examples and shared experience.

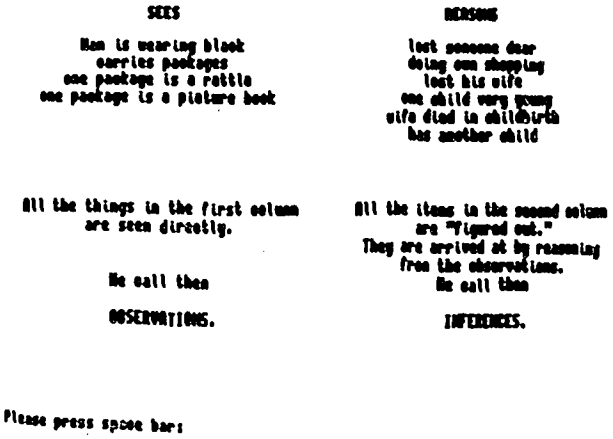


Figure 3

Introduction of terms, Observation and Inference.

The next sequence in the program concerns counting the elapsed time interval between lightning and thunder and calculating the distance between the observer and the lightning strike, the student having to answer questions about observation and inference in this connection. This sequence, however, is not needed by all users. If someone shows no difficulty going through the first "Sherlock Holmes" sequence, then the lightning and thunder activities are bypassed.

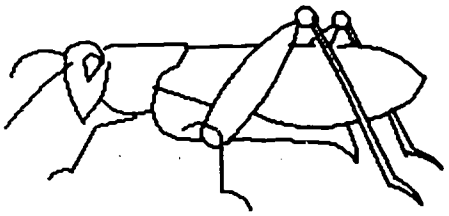


Figure 4

Graphics for activity on counting grasshoppers.

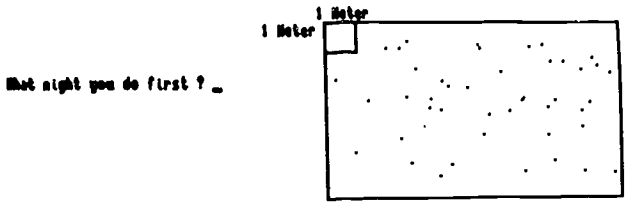


Figure 5

Opening question for grasshopper problem.

The next component of the program involves statistical inference. The student is presented with a large field containing many jumping grasshoppers (Figs. 4-5). The student is told that it is impossible to count all the grasshoppers directly, and he is invited to suggest alternate strategies for getting some idea of how many grasshoppers there are in the field. Thus approximation and sampling come into this activity. The student is led, though a series of questions, to the idea of counting the grasshoppers in a one meter square. The students then count directly, watching the one meter square with the jumping grasshoppers (Fig. 6). The number of grasshoppers in the square fluctuates slightly. The program checks their value and gets them to count again if they are far off. Students then determine the total area of the field, and obtain an estimate of the number of grasshoppers in the entire field. Again, questions are asked about what is an observation and what is an inference in this sequence.

There is a final sequence providing further exercises in discriminating between observation and inference. We also show the student that not everything we do can be classified as either an observation or inference. For example, defining a concept or inventing a name do not fall in either category.

Observations of Usage

The modules developed under this grant, and under a FIPSE grant concerned with public understanding of science, have all been tested in a variety of environments. The principal testing environment has been public libraries. We have found this to be a very useful environment, not only to identify conceptual weaknesses in the programs, but also to be sure the programs are motivationally strong, holding viewers even in a library environment where there is no pressure

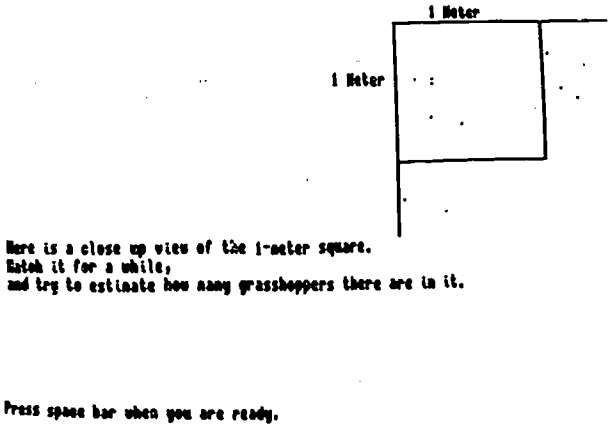


Figure 6

Exercise on counting a fluctuating number of grasshoppers.

to stay at the display.

This program has gone through a number of revision stages after such testing. It now works well with a wide group of individuals. We are also using it as one of the programs in another research program involving the behaviour of groups in using computer based learning material. In this case we videotape groups employing the material, capturing their conversation, emotions, and key pushes at the computer. This, too, has been useful in understanding how the program works and in leading to additional revisions.

Plans

The program currently runs on its developmental hardware, the Terak 8510/a. The choice of delivery hardware will be made by the Educational Technology Center and the distributor. We are currently conducting discussions with several companies about the possibility of having these materials available commercially.

At the moment, users with Terak 8510/a's can obtain copies of the program at our cost. An order form is available from the authors. Currently the program is not available on other personal computers, although segments run on the IBM Personal Computer in connection with another program.

DOES USE OF MICROCOMPUTERS IN JUNIOR HIGH SCHOOL INCREASE PROBLEM SOLVING SKILLS

Barbara Kurshan
Joyce Williams
Nancy Healy

Hollins College
Hollins College, VA 24020

ABSTRACT

This is a study to determine if the use of the microcomputer increases problem solving ability of seventh grade students. Two seventh grade classes from similar schools in Roanoke, Virginia were selected for the study. The first group was exposed to introductory computer literacy/computer programming activities for an entire year in a microcomputer lab. The second group did not have a computer lab in the school. The preliminary results indicate that students exposed to computers show increased problem solving ability. Hopefully, this study will encourage others to explore the benefits of the computer as a tool for increasing problem solving skills.

Introduction

"Children learn by doing and by thinking about what they do" (Papert, 1980, p.161). This process of doing and thinking should ultimately provide children with the ability to solve problems. The introduction of the microcomputer into the learning process enhances the "doing" area of learning. It gives children the "power" to view and solve exciting problems. The skills used to solve problems are perhaps enhanced by the use of the computer. Students appear to be able to grasp the true significance of broader problems. They are able to apply algorithms that in the formal rather than experimental setting are lost to the learner.

The assessment of the impact of microcomputer use on the problem solving abilities of students has diverse benefits. Educational decision makers can use the data for substantiating the need for computers in the school. Curriculum designers could redesign portions of courses that teach problem solving and include a greater emphasis on computer learning. The classroom teacher will hopefully be more inclined to use the computer for project design, creation and implementation. In general, the educational community certainly realizes and enthusiastically embraces the need for the computer in the classroom because of its inevitable impact on society. However, if this study and future research can give some evidence to the increased problem solving ability that students gain from "doing" then the issues concerning the microcomputer in the classroom will perhaps have a central focal point. This study is an initial attempt to begin to form that base of data to answer the question "Does the use of the microcomputer for learning increase problem-solving ability?"

Background

The emphasis of the research concerning the effect of classroom computer use on student performance has focused on comparing CAI (Computer-assisted instruction) to traditional instruction and the effects of computer programming by students on problem solving skills. The major emphasis of research concerning general computer use has investigated the effect of CAI use on achievement when compared to traditional instruction (see for review Dence[2], Edwards, Norton, Weiss and Dusseldorp[3] Forman[4]; Jamison, Suppes and Wells[10] and Kulik, Kulik and Cohen[12]). Research that has investigated the effect of CAI and traditional instruction has generally shown that the combination of CAI and traditional instruction is the most effective and requires less instructional time than traditional instruction.

The reviewers of the CAI vs traditional instruction studies generally support the effectiveness of classroom computer use. Edwards, Norton, Taylor, and Dusseldorp[3], for example concluded as a result of their study in elementary schools, traditional instruction supplemented by computer based instruction was more effective than traditional instruction along. Jamison's et al.[10] survey of the effect of CAI studies found that disadvantaged elementary school students appeared to show the most achievement gain when using CAI. Further, Jamison's et al.[10] concluded that CAI was most effective when used as a supplement to regular instruction and that the instruction time was less. Edwards' et al.[3] and Jamison's et al.[10] findings indicate that using CAI in conjunction with regular classroom instruction improves achievement, that students take less time to learn the material and students that are less academically prepared and from lower socioeconomic appear to benefit more from computer use.

Dence[2], Forman[4], and Kulik's et al.[12] reviews present a current perspective of the state of

computer use on student achievement and identify outcomes of computer use that are very similar. In accordance with the aforementioned researchers, Dence[2] reported that CAI students that receive CAI and traditional instruction obtain higher scores than those students who receive only CAI or only traditional instruction. Dence also reported that CAI students appear to have a greater retention of material than students that were taught only by the traditional method. Students that had prior familiarity with CAI or subject matter benefitted more when using CAI and students with initially low levels of achievement tend to show greater test gains (pretest and posttest)[2]. Similar conclusions were made by Forman[4]. It appears that regardless of the age of the student or type of hardware used students tend to show improvement in achievement scores. Further, it seems that students that have prior experience with the computer or subject tend to benefit even more from computer use.

The general conclusion drawn from the literature related to the issue of CAI vs traditional instruction is that CAI, when used as a supplement to traditional instruction, does produce greater achievement. Further, there appear to be factors other than the delivery method that influence achievement when CAI is used in schools. For example, the socioeconomic factor introduced by Jamison et al.[10]; the prior familiarity factor introduced by Dence[2] and the ability factor, addressed by Jamison et al.[10,2] and Forman[4], seem to influence the amount of achievement gain and instruction time.

Investigations of the effects of computer programming by students on problem solving skills (Johnson and Harding[6], Milner[13], Ronan[14], Wilkinson[17], and Holoien[9]) have also been conducted. Research has shown that students learn the content better when they write and run their computer programs (Foster[5], Johnson and Harding[6], Milner[13], Holoien[9], Ronan[14], and Wilkinson[17].) Foster[5], for example, investigated how students' use of computers and flow charts effect their problem solving ability. Sixty-eight eighth graders were placed in four treatment conditions: (1) use neither computer nor flow chart; (2) use flow charts only; (3) use computers only and (4) use computers and flow charts[5]. Over a period of twelve weeks each student was provided with 24 tasks that required both computer and non-computer solutions. Foster found that the third condition (i.e. only used computer) had significant mean differences on processing hypothesis, identifying a pattern, and selecting relevant data. The data also showed that group 2 and 4 performed better than the groups that used neither the computer nor flow chart. Milner[13] and Ronan's[14] findings are very similar to Foster's[5]. That is, students that were taught programming and given problem solving tasks to perform showed greater achievement gains than those students that did not use the computer.

Description of Study

The research comparing CAI to traditional instruction and the research investigating the effect of computer programming on students' problem solving behaviors indicate that general computer use enhances students' achievement and problem solving performance. The present study while continuing in the same tradition as those previously cited, differs on several specific features. First, the treatment consists of only one group of students being exposed to the computer in math class. However, the students were not trained in a specific computer skill nor were they given specific CAI material to use. They were given a variety of computer experiences including games, programming, CAI and simulations. The control group did not use computers in their math class.

The second feature of this study is data obtained from school records of eighth grade students in both schools. Third, problem solving behaviors were assessed by students' performance on the problem solving subtest of the SRA Achievement Series test and the Hartman Test of Causal Reasoning. Finally, prior familiarity was manipulated by assessing students' general use of computer through their ratings on the Prior Exposure to Computers Index (Anderson, et al.[1]). It should be noted that for the purposes of this study, general computer exposure is defined as using CAI, gaming and simulation, computer programming, and experience with arcade games.

The Prior Exposure to Computers Index was used for the reasons of ecological validity; it was reasoned that while general computer use has been used to establish the power of a variable, validation in a natural setting with students of similar SES and intellectual ranking, and the Prior Exposure to Computers Index would strengthen the conclusion of the difference in performance being due to general computer use in the math classroom. Thus, the predictions based on the research are that: (1) students that are exposed to computers in their math classroom, regardless of their prior exposure to computers, will show significantly higher problem solving scores on the problem solving measures; (2) students that have had little or no prior exposure to computers and are exposed to computers in their math classroom will show significantly higher problem solving scores on the problem solving measures than no prior exposure students that did not use the computers.

Procedure

The experimenters interviewed school officials in the spring of 1982 to receive permission to proceed with the study and to receive advice on choosing a control school. The experimental school was chosen by the fact that it was the only junior high school in the city where all students within a grade would use computers. The control school was chosen because it was more similar to the experimental school than any other junior high school in the city. In the fall of 1982, the principals, teachers, and guidance personnel at the two schools were involved in the study.

Standardized test scores and other relevant data were gathered from school records by the experimenters and recorded on the Check-Off Sheet for Computer Study. They were assisted by two Hollins College student assistants. Data

includes scores for reading, math, problem-solving and EAS - Educational Ability Series Tests.

The curriculum at Woodrow Wilson does not, at the present time, include instruction using computer instruction if funds become available. Students at Breckinridge were offered the opportunity to enroll in microcomputer/math classes during the 1982-1983 school year. A follow-up study of standardized test scores for the spring of 1983 is planned for both the experimental and control groups.

Selection of Students

Students selected for this study come from the present eighth-grade classes at two of Roanoke City's six junior high schools, Breckinridge and Woodrow Wilson. These two schools have similar socioeconomic profiles.

The population at both schools can be described as middle-income but with some lower-income and some higher-income families. Single family homes predominate the neighborhoods from which these schools draw students, but both school populations include students who live in federally-funded housing projects.

The city's school population is 20% black and almost 80% white. At Breckinridge, almost 23% of the students are black and 77% are white. At Woodrow Wilson, almost 26% of the students are black and almost 75% of the students are white. Less than 1% of the students from either school belong to other ethnic or racial groups.

Both schools consider their student population very stable, with transient students not a problem. The ability levels and achievement levels of the total student population within the two schools are similar to within several points on standardized tests.

The experimental group consists of the students at Breckinridge Junior High School where computers were used in math classes. The control group consists of students at Woodrow Wilson Junior High School were there were no computers in classrooms.

Students who transferred into or out of either school were excluded from this study.

Selection of Tests

Several testing instruments were used in this study. To test for problem-solving ability, the problem-solving subtest of the Mathematics portion of the SRA Achievement Series were used (SRA,1980). Sixth-grade scores were used as a pre-test; seventh-grade scores were used as a post-test. An additional measure of problem-solving ability was the Hartman Test of Causal Reasoning, (Hartman[7]). To determine the level of prior exposure to computer use, questions were formulated from the Anderson, Klassen, Hansen, and Johnson[1] study.

The SRA Achievement Series was used for several reasons. It has previously established levels of reliability and validity. Since students take these tests every spring as part of the school's regular testing program, they cost students no out-of-class time. They provided pre-test information which could not have been obtained otherwise. The Math Portion of the SRA Achievement Series included a subtest specifically aimed at the area of problem-solving.

The Prior Exposure to Computers Index (Anderson, et al.[1]) was used as the measure of students previous exposure to computers. It was taken from a study of affective and cognitive effects of microcomputer based instruction.

The Hartman Test of Causal Reasoning was chosen as an additional measure of problem-solving ability. It is not a mathematical test and requires more verbal reasoning skills than mathematical reasoning skills. It requires 55 minutes of class time to administer. The instrument is a multiple-choice Test of Multivariable Causal-Logical Competence, constructed by Cheryl Hartman [7] to assess the four forms of reasoning (i.e. Form I: the ability to validly induce an equivalence causal relationship, Form II: the ability to validly induce a multiple sufficiency relationship, Form III: the ability to validly induce a multiple necessity relationship and Form VI: the ability to validly induce that a data contradiction can only be resolved by invoking a hypothetical variable; Forms IV and V of the model were not studied due to pragmatic limitations.)

Method of Testing

The SRA tests were administered by homeroom teachers within each school. Students took these tests in the spring of 1981 and 1982 as part of the regular school program. Teachers giving these tests received standardized instructions for administering them.

The Hartman Test of Causal Reasoning will be administered to 30 students from each school population. Students will be chosen randomly within male and female groups. These students will be tested in groups of five with the experimenters serving as the testers. This setting should approximate the individual testing done in Hartman's research.

Analysis

T-tests were done on the groups (school, prior exposure, and sex) the means of GainEAS scores, EASproblem solving scores and the Hartman Problem Solving test scores were compared by group. The probability level of ($p<.10$) was set. Even though this was an expofacto experiment, a 90% chance of computer use in the classroom affecting students' problem solving ability is important.

EAS Gain Score Analysis (GainEAS)

The T-tests comparing the "GainEAS" by school and sex were not found to be significant at the .10 level. There was a notable difference in the "GainEAS" means scores in favor of Breckinridge. However, the small sample sizes and large variability between the scores may possibly have influenced the findings of no significance (Table I).

There was a significant ($p<.074$) between the mean "GainEAS" score of low exposure Breckinridge students ($X=5.8983$, $SD=9.234$, $N=59$) and the mean "GainEAS" score of low exposure Woodrow Wilson students. These results indicate that the Breck-

inridge low prior exposure to computers, students benefitted more, in terms of "GainEAS", than any of the other students in the study.

Table I
GAINEAS

Group	Mean	SD	T	Prob.
Schools				
Breckinridge (N=93)	4.8280	8.872	1.26	.209
Woodrow (N=104)	3.3269	7.708		
Sex*				
Girls at Breck. (N=53)	5.7736	8.617	1.21	.230
Girls at Woodrow (N=59)	3.9153	7.544		
Boys at Breck. (N=40)	3.5750	9.156	.55	.587
Boys at Woodrow (N=40)	2.556	7.936		
Prior Exposure				
High-Breck. (N=34)	2.9706	7.998	-0.43	.672
High-Woodrow (N=8)	3.8750	4.518		
Low-Breck. (N=59)	5.8983	9.234	1.81	.074*
Low-Woodrow (N=96)	3.2813	7.930		

* Sex of some subjects were unknown.

EAS Problem Solving Gain Score (GainPS) Analysis

Significant ($p<.10$) differences were not found in the mean "GainPS" scores of groups (school and prior exposure) (Table II). There was, however, a significant difference ($p<.002$) shown between the "GainPS" in high exposure students at Breckinridge ($X=2.3056$, $SD=.208$, $N=36$) and high exposure students at Woodrow Wilson($X=-1.000$, $SD=1.549$, $N=6$). The high-exposure students at Breckinridge apparently gained in problem solving skills over the year. This gain may possibly be attributed to these students use of the computer in the classroom. There was also a significant ($p<.04$) difference in the problem solving gain scores between the boys at Breckinridge ($X=2.5950$, $SD=4.150$, $N=40$) and the boys at Woodrow Wilson ($X=.8000$, $SD=3.436$, $N=40$). It is apparent that more boys at Breckinridge increased their EAS scores than boys at Woodrow Wilson. As mentioned earlier, the level of significance may have been influenced by the amount of variability between the scores and the small sample sizes. Nonetheless, it appears that students that were exposed to computers in the classroom did show improvement in their problem solving scores on a standardized measure.

Table II
GAINPS

Group	Mean	SD	T.	Prob.
Schools				
Breckinridge (N=90)	2.33	4.224	1.22	.225
Woodrow (N=90)	1.6111	3.714		
Sex				
Boys (N=80)	1.875	3.890	0.86	.390
Girls (N=100)	2.200	4.060		
Boys Breck. (N=40)	2.5750	4.150	2.08	.041*
Boys Woodrow (N=40)	.8000	3.436		
Girls Breck. (N=50)	2.140	4.314	-0.15	.883
Girls Woodrow (N=50)	2.2600	3.832		
Prior Exposure				
Low-Breck. (N=54)	2.3519	4.274	.78	.423
Low-Woodrow (N=84)	1.7976	3.757		
High-Breck. (N=36)	2.3056	4.208	3.50	.002*
High-Woodrow (N=6)	-1.000	1.549		

Hartman Problem Solving Test Analysis (PSScore)

The data from the Hartman test analysis is not complete. However, preliminary analysis show that there is no significant difference between the performance of girls and boys on this test. The analysis does show an unexpected significant difference between the mean test score of the two schools. That is, it appears that the control school (Woodrow Wilson) students performed better on the Hartman test than the Breckinridge students who were exposed to computers in the classroom. However, it is again noted that these data are incomplete and inferences made from it are not reliable. This test is being repeated for Spring, 1983 data.

Discussion

The data support to a degree the beginning assumption that students that have been exposed to computers in the classroom will show high problem solving scores. The data even though incomplete indicates that high exposure students that have been exposed to computers in the classroom tend to increase their problem solving scores. There is no significant evidence that students that have had low exposure to computers and are exposed to computers in the classroom perform better on problem

solving measures. However, there is evidence that students that have little prior exposure to computers and are exposed to computers in the classroom do improve their over all intelligence scores. Perhaps upon completion of the Hartman analysis the picture of students problem solving performance will be clearer.

Conclusion

The use of computers in education will continue to grow at an even more rapid rate than today. Educators need to know that the computer is more than a "fun" way to learn. If computer use does increase problem solving ability then the designers of curriculae should incorporate this factor into learning programs. However, it is difficult to satisfactorily design a program to teach problem solving skills. Therefore, if the computer can be used and one of the inherent benefits, whether formally identified or imbedded in the "nature of the beast", is an increased problem solving ability, then learning with micros is certainly ideal.

References

1. Anderson, R.E., Klassen, D.L., Hansen, T.P., & Johnson, D.C. The affective and cognitive effects of microcomputer based science instruction. Educational Technology Systems 1981, 9, 329-35.
2. Dence, M. Toward defining a role for CAI: A Review. Educational Technology, 1980, 20, 50-54.
3. Edwards, J., Norton, S., Taylor, S., Weiss, M., & Dusseldorp, R. How effective is CAI? A review of the research. Educational Leadership, November 1975, 33, 147-153.
4. Forman, Denyse. Search of the literature. The Computing Teacher, 1982, 37-49.
5. Foster, T.E. The effect of computer programming on student problem solving behaviors in eighth-grade mathematics (Doctoral dissertation, University of Wisconsin, 1972). Dissertation Abstracts International, 1973, 33, 4239A.
6. Harding, R.D. Computer-aided teaching of applied mathematics. International Journal of Mathematical Education in Science and Technology. 1974, 5, 447-455.
7. Hartman, Cheryl W., The construction and empirical investigation of a model of multivariable causal logical competence, Appendix, Roanoke, Va., 1982.
8. Hatfield, L.L., & Kieren, T.E. Computer-assisted problem solving in school mathematics. Journal for Research in Mathematics Education. 1972, 3, 99-112.
9. Holoien, M.O. Calculus and computing: A comparative study of the effectiveness of computer programming as an aid in learning selected concepts in first-year calculus. (Doctoral dissertation, University of Minnesota, 1970) Dissertation Abstracts International, 1971, 31, 4490.
10. Jamison, D., Suppes, P., & Wells, S. The effectiveness of alternative instructional media: A survey. Review of Educational Research, 1974, 44, 1-61.
11. Johnson, D.C. Programmed learning: A comparison of the school mathematic study group programmed and conventional textbooks in elementary algebra (Doctoral dissertation, University of Minnesota, 1965). Dissertation Abstracts Internationa, 1966, 26, 5294.
12. Kulik, J.A., Kulik, C.L.C., & Cohen, P.A. Effectiveness of Computer Based College Teaching. Educational Technology, 1981, 307-318.
13. Milner, S.D. The effects of teaching computer programming on performance in mathematics. (Doctoral dissertation, University of Pittsburgh, 1972). Dissertation Abstracts International, 1973, 33, 4183A.
14. Ronan, F.D. Study of the effectiveness of a computer when used as a teaching and learning tool in high school mathematics. (Doctoral dissertation, University of Michigan, 1970). Dissertation Abstracts International, 1971, 32, 1264A-1265A.
15. SRA Achievement Series, Forms 1 & 2. Science Research Associates, Inc., 1980.
16. Taylor, Robert P., editor, The Computer in the School; Tutor, Tool, Tutee, Teachers College Press, New York, 1980.
17. Wilkinson, A. An analysis of the effect of instruction in electronic computer programming logic on mathematical reasoning ability (Doctoral dissertation, Lehigh University, 1972). Dissertation Abstracts International, 1973, 33, 4204A.

DIVERGENT ANSWERS TO THE QUESTION, "WHERE SHOULD COMPUTER EDUCATION DOLLARS BE SPENT?"

Arthur Luehrmann
Computer Literacy

Eric F. Burtis
President, Centurion Industries, Inc.

Beverly Hunter
Human Resources Research Organization

ANSWER 1:

Not on General Purpose Computers at the Elementary Level. All this emphasis on general purpose computers is taking elementary school dollars away from where they belong: teaching the basic skills of reading, writing, and arithmetic. What good are computer skills and knowledge to a kid who can't even spell or add? Save the money and invest it in effective systems for teaching basic skills. -- Eric F. Burtis.

ANSWER 2:

On the Social and Ethical Issues of Computer Use. Before committing vast sums to training a generation of programmers, we should be certain that all students know how computer use, proper and improper, can affect the individual and the society. Mere technical expertise is not the answer. -- Beverly Hunter.

ANSWER 3:

On Teaching a New Basic Skill: Computing. Learning to use a computer is learning a new way of writing and thinking. People who have this skill can solve more complex problems than others, manage information better, and get better jobs. Schools should provide these new skills for the same reasons they teach other basic skills. -- Arthur Luehrmann.

An Evolving Model for Providing Computer Education for Gifted Children

Mary Crist, Chair
The Ames Hill Center for Gifted Children
Wilbraham and Monson Academy
Wilbraham, MA 01095

ABSTRACT

The Ames Hill Center for Gifted Children, a part of the Wilbraham and Monson Academy, serves 340 students (ages 3-16) from the Greater Springfield metropolitan area in western Massachusetts, and northern Connecticut through the Afterschool, The Saturday School, and two summer sessions.

Although there often appears to be little agreement as to the true nature of giftedness, this presentation will discuss five specific characteristics of the gifted child that appear to be critical to learning computer programming. Consideration will also be given to the influences of I.Q. and chronological age. A challenge to educators is issued.

The computer education program at Ames Hill consists of four programming languages. The Logo language serves to introduce students with no previous computer experience into the non-threatening world of computers. BASIC, PASCAL, assembly language, and FORTH guide students into creative areas of computing--problem definition and solution through algorithm definition. The pros and cons of the programming languages we have experienced will be presented.

Observed characteristics, including individual learning styles of gifted children in computing courses, will be described. Factors, such as the ability to work with a partner, tolerance for frustration, and the needs for exploration and limited structure will be examined. The presentation of case studies of students in Logo will provide additional information regarding processes used in problem solving and the relationship between (1) age and level of achievement, and (2) selection of goals.

The cost of microchips continues to decline, and as it does, the economics dictate that a whole new set of tools will become available. We will examine these tools, including speech, speech recognition, robotics, and others, and their anticipated impact upon the education of gifted children.

Pat Semmes
Department of Computer Science
Trinity University
San Antonio, TX 78284

Elaine Henshon
The Ames Hill Center for Gifted Children
Wilbraham and Monson Academy
Wilbraham, MA 01095

Training University Faculty in the Use of Computer Graphics

Richard G. McGinnis
Bucknell University

ABSTRACT

Interactive computer graphics is a powerful means of presenting and manipulating complex data in visual form. The technology of this field has grown rapidly in the last decade; and computer graphics is now widely used in the areas of engineering and science that require design, analysis, and the presentation of digital data. Other, less well known applications have been developed in non-engineering disciplines: digital data processing in linguistic studies, art and dance applications, computer-assisted musical score generation, demographic and geographic mapping, medical x-ray image processing, and social science statistical displays.

Recent efforts to incorporate this technology into the curricula of higher education have been largely restricted to the engineering disciplines. There is, however, enormous potential for the application of computer graphics into other curricula at both the undergraduate and graduate levels. For example, in the social sciences, computer mapping routines could be used to show the distribution of various demographic characteristics such as income levels, ethnic populations, contours of percents of political party registrations, contours of the incidents of various diseases, etc. In art, students could use computer graphics packages to experiment with various shapes and shadings to produce different types of drawings; while chemistry students could use graphics to investigate the molecular structures of various compounds.

In 1982, Bucknell University received a grant from the Exxon Education Foundation for the purpose of expanding Bucknell's interactive computer graphics capability and its curricular impact to appropriate disciplines in the sciences, social sciences, and humanities. Since computer graphics is a relatively new teaching tool and many faculty members have little, if any, knowledge about its potential applications in undergraduate instruction, the project emphasized faculty training and curricular revision.

In the first phase, the project director met with the departments on campus to explain the characteristics of computer graphics and to identify potential applications of computer graphics within the various disciplines. Next, interested faculty members submitted proposal indicating how they would like to use computer graphics within various disciplines. Next, interested faculty members submitted proposals indicating how they would like to use computer graphics in courses, and from these proposals software needs and equipment needs were identified. The third phase, faculty training and course development, occurred during the summer when the twenty-two participants were given intensive instruction, including an overview of computer graphics concepts, the use of computer graphics devices, "hands on" experience with available software packages, and discussions of appropriate ways to improve teaching effectiveness through the use of computer graphic technology. Following the instruction period, the faculty participants developed the teaching materials necessary to integrate their proposed graphics applications into their courses. Implementation occurred during the 1982-83 academic year.

Case studies of two of the faculty participants are presented. The first is that of an English/Theatre professor who had no prior computer experience and who wanted to use computer graphics as an aid to teaching scene design. One of the difficulties of teaching scenic design is that students with a keen interest and talent for design often have poor rendering skills and little or no understanding of perspective drawing techniques; and thus, they cannot accurately visualize their ideas on paper. However, computer graphics can lift theatre design classes from the realm of drawing and composition into what they are intended to be: courses in scenography.

The second case study presented is that of an economics professor who had previous computer experience but none with computer graphics. Her project involved the development of a program to produce "bulging" pie charts that could be used for econometric analyses.

The project was very successful in stimulating faculty and student interest in computer graphics. During Fall, 1982, there were over 14,000 programes run (total student enrollment at Bucknell is 3,500) using computer graphics, and the level of activity should increase as faculty members implement additional uses of graphics in their courses.

PANELISTS:

Daniel C. Hyde
F. Elaine Williams
Jean A Shackelford
Bucknell University
Lewisburg, PA 17837

Recommendations for Programs in Computing at Small Colleges

John Beidler, Chair
University of Scranton
Scranton, PA 18510

ABSTRACT

An ad hoc committee of the ACM Education Board has been formed to update and revise the curriculum recommendations for small colleges that was published in 1983. Since a number of fine curriculum recommendations have been published by the ACM and other organizations, this committee plans to develop its report as a consulting/planning document that takes into consideration the special environmental and resource problems that many small colleges must face.

The committee will complete its report before the end of 1983. At this time the committee will present a preliminary draft of its report. The audience will be encouraged to respond to the report and provide their input to the committee.

PARTICIPANTS

Richard Austing
University of Maryland
College Park, MD 20742

Lillian Cassell
Goldey Beacon College
Wilmington, DE 19899

COMPUTERS AND QUANTITATIVE METHODS: HEALTHY FOR THE HUMANITIES?

by Rudy S. Spraycar

Data Processing Department
United States Fidelity and Guaranty Company
Baltimore, Maryland 21203

Abstract

The advent of the computer has enhanced the ability of the humanist to apply the quantitative and statistical research methods that have been a mainstay of the natural and social sciences. Controversial in nature, these techniques have at times been misused. Further, a solid theoretical foundation for quantitative approaches to research in the humanities is still to be sought.

Introduction

Since C. P. Snow first pointed out "how very little of twentieth-century science /had/ been assimilated into twentieth-century art"[1] or into the "Literary Culture" in general, and W. H. Auden wrote, "Thou shalt not sit / With statisticians...,"[2] many humanists have made great strides in the learning of science and technology for which Samuel M. Hines, Jr., has called.[3] Indeed, C. E. Kaylor, Jr., went so far as to argue that "much of the research in the humanities has now become at least quasi-scientific,"[4] and if this is so it can be laid in great measure at the door of the computer. Perhaps no technological development has had so great an impact upon research in the humanities as the computer has had. Apart from facilitating the preparation of such traditional research tools as bibliographies, indices, and concordances, the computer made possible the proliferation of quantitative and statistical analyses of the matter of the humanities.

This is a salutary counterweight to the efforts to fill such perceived vacuums in disciplines outside the humanities as the need for ethics and values, as Hines has stressed;[5] it is a hopeful and healthy sign that humanists are now methodological borrowers from as well as lenders to their scientific colleagues.

The Problem

Nevertheless, this trend has exacted a considerable price. Bowman L. Clarke recently observed that humanists' welcoming new methods and technologies in spite of their colleagues' rejection of such "Faustian" behavior has polarized some disciplines; he offers the example of the gulf between "humanistic psychology" and "experimental psychology."[6] Giles Gunn would add the extremes among historians represented by "the computer programming of the cliometricians" on the one hand, and the studies of cultural historians of the traditional school, on the other; Gunn also finds European structuralism "scientistic to the core."[7] Far less polemical is Morton W. Bloomfield's distinction between the "part of the humanistic process /that/ is obviously analytic and scientific," concerned with knowledge "from the outside," and, on the other hand, the concern with "inward experience,"[8] as he recasts the commonplace distinction between "objective" and "subjective" elements of humanistic study.[9] It is the former with which the present paper is concerned, drawing examples chiefly from literary research. After all, Monroe C. Beardsley has remarked that "in so far as the student of literature is interested in... describing or interpreting what he finds, he relies upon plain (though by no means simple) empirical methods."[10]

Eric Weil puts the interdisciplinary borrowing of empirical tools in perspective when he reminds us that "psychoanalysis, statistics, and structural analysis provide the humanist with new tools he must employ simply because they are there. These tools give him the opportunity to look at his material from new perspectives.... What we have...are new auxiliary sciences.... Auxiliary in relation to the humanities.... But does it not then follow that the humanities should have their own kind of 'scientificity,' besides and above that of these sciences...?"[11] Or as Clarke puts it, "What is needed is a new philosophic stance /that/ must see scientific methodology for what it is, merely a sharpening of man's powers of accurate observation and logical deduction.... On this depends the future of the humanities."[12]

But what of the peculiar problem raised when a humanist borrows only the most rudimentary of the quantitative tools and methodologies developed by the natural sciences and in some measure appropriated by the social sciences? Here, the presence of a computer printout only seems to enhance the objectivity of the results; the adage "Garbage in, garbage out" seems apt. In other

words, the dangers of the American version of the two cultures really lie less with the stolidly resistant humanist whose knee-jerk reaction against computers and quantitative methods may well be fundamentally anti-intellectual than in the ignorance, however tolerant or enthusiastic, of the bliss humanist who applies with apparently elegant simplicity a methodology borrowed from our scientifically-trained colleagues to an age-old humanistic crux or conundrum.

Pseudo-Science

The study of Classical and medieval epic offers an example that illustrates some generalities about hasty and literally "undisciplined" inter-disciplinary borrowings. It is to a would-be scientific impulse that one may attribute the attractiveness of an "acid test" to establish simply and conclusively the provenance of a troublesome text. I refer to the relatively well-known "oral-formulaic" theory of Milman Parry and Albert Bates Lord,[13] originally a rather subtly conceived and carefully qualified hypothesis that the study of Homeric style may be illuminated by the living oral epic tradition in Yugoslavia. In latter days, however, the attractiveness of an easy, quantitative mode of analysis for troublesome Classical and medieval texts, inspired less by scientific method than by a credulous misunderstanding of what a "living laboratory" of poetic composition might signify to a "man in a white coat," does little justice to Professor Milman Parry's original genius. To be sure, it is a radically attractive notion that one might, simply by counting the number of poetic phrases repeated elsewhere in a "literary" text, adding them up, and dividing by the number of phrases in the whole sample, thus calculating a "percent formularity" as a measure of the repetitiveness of the text, demonstrate incontrovertably that a given poem--say Beowulf or the Chanson de Roland--was composed by illiterate means, despite the common-sense testimony of the text's survival in written form.

But some of the followers of Professors Parry and Lord have been plagued by such methodological difficulties as the labelling as "statistical analysis" of what should be called "raw data": numbers generated by elementary arithmetical operations like counting or adding up and dividing. Indeed, few studies of this literary problem have mentioned even so fundamental a qualification as the "margin of error" resulting from the sampling technique that we have come to expect from the purveyors of public opinion polls. Similarly, the discounting of a wide disparity in the relative lengths of two samples of poetry analyzed for repetitive diction (without apparent regard for the probabilistic principle that the longer anyone, poet or not, speaks, the greater is the likelihood of repetition), flies full in the face not only of mathematics but of everyday experience as well.[14] Further, to find, for example, in a disparity of five percentage points between the degrees of repetitiousness of two poems of unequal length grounds for making a determination whether a text was composed orally or in writing betrays a misplaced faith in crude arithmetic.[15] Here one might wish to be reassured that a five percent difference in "percent formularity" cannot be attributed to the margin of error.

Among studies in this area, there are few instances of collaboration between humanists and statisticians. To be sure, some recent dissertations have been written by scholars with serious training in both the humanities and in, for example, statistics.[16] Though such scholars are rare, and their works make unprecedented and often nearly impossible demands on their humanistic readers, they nevertheless represent a healthy trend that will in time contribute greatly to a solution of a part of the problem.

In the meantime, however, while the ranks of such interdisciplinary scholars are thin, other problems are arising as less well prepared humanists attempt to develop new methods on their own. One of the worst consequences of this activity is its effect on the humanities' credibility in the intellectual community at large. Of much greater destructive potential, however, are the consequent resentments and divisiveness among humanists.

Methodological Caveats

What deserves our careful consideration, then, is less our attitude toward new research tools in general than our profound need to carry over along with these tools the methodological caveats and restrictions that responsible scientists place on their use. The real problem in need of resolution and requiring the concerted efforts of us all is not a conflict between technological or quantitative approaches and the matter of humanistic study; it is the disregard for the very intellectual apparatuses without which the scientists' tools are little more than toys, that can be inimical to traditional humanistic concerns.

For example, a bulwark of scientific method is the formation and testing of hypotheses. But John Reichert has pointed out that "one of the tests of a scientific hypothesis is its predictive capacity. To the extent that it predicts well other events beyond the event it was designed to explain, its validity is established. This aspect of hypothesis testing points up nicely one of the fundamental features of criticism.... Our critic in the role of interpreter or teacher is not concerned with the discovery of general laws that will apply beyond the work. His target is an understanding of the individual literary event in its uniqueness. In this respect his task is somewhat like the historian's as it was conceived by philosophers like R. G. Collingwood and Michael Oakeshott, both of whom stressed the uniqueness of historical events and the need to "get inside" them."[17]

Or, in Frank Kermode's more succinct terms, "Certainly science is about reality experienced as repetitive, and art about reality experienced as constituted of unique events...."[18] Of course as Eric Weil has observed, when the humanities, or

more usually the social sciences, examine non-unique phenomena, their methods approach those of the natural sciences: "The social sciences are situated on the same terrain as the humanities--that of history. They are not interested in individual acts as such, but they deal with such acts globally considered. They do not go into the private convictions of John Doe; they look at the group of which he is a member or at him as representative of this group. Their method is statistical; their purpose is to discover necessary relations within a certain series...."[19]

But because the oral-formulaicists have attempted to use statistics and are interested precisely in the relation of a poem's style to that of others in a "series," the methodologies of the natural sciences and consequently their inherent strictures would seem to obtain here; Northrup Frye sums up the latter well: "The physical sciences at least are not simply descriptive, but are based on prediction as well.... It is not the experiment but the repeatable experiment that is the key to the understanding of nature in the physical sciences, and the repeatable experiment is what makes prediction possible, and gives to science a prophetic quality."[20] The predictive value of a hypothesis is at once tested and established by its replicability: as Bruce A. Beatie insists, "only when an hypothesis has been confirmed by repeated experiments is it...provisionally accepted as a valid interpretation of observed data.... The original Parry-Lord hypothesis, however, remains unconfirmed by repeated experiment...."[21] As I have argued elsewhere, not only have the followers of Parry and Lord repeatedly modified their methodologies[22] in studying poetry in various languages, undermining the hypothesis' claim to universal validity, but my own attempts[23] to test, in the very Serbo-Croatian tradition upon which the hypothesis is based, its prediction that no literate poet can write as formulaically as an oral poet can compose, has cast further doubt on the usefulness and reliability of the methodology: in fact I found no significant quantitative distinction between oral and written Serbo-Croatian verse based on the number of repeated phrases that they exhibit.

That the replicability or predictive power of so explicitly quantitative an hypothesis was left untested for decades is certainly cause for concern. But I am troubled far more by the question whether scientific and quantitative methodologies, even when borrowed by humanists so as to preserve fully their explanatory powers, are finally suited to the analysis of poetry.

Random Behavior

Of special difficulty here is the dilemma faced by those tackling the essentially quantitative problem of the degree of repetition as a factor of poetic style. "Degree" must be measured, and it is no mistake to borrow here from our colleagues in the natural and social sciences, for whom measurement is often of paramount concern. But statistical analysis evidently requires the assumption at some level of random behavior, whether that of the agitated vibrations of molecules or of atomic particles or that of the collective human expression of political opinions. Northrup Frye ties the predictive power of the natural sciences and of their adjunct quantitative methods to the unconscious character of the phenomena observed: "Where the phenomena are unconscious or where the units involved are small and numerous, like atoms, molecules, or cells, so that there is no practical difference between the highly probable and the certain, the language of science is primarily mathematical. From the natural sciences we move toward the social sciences, where the phenomena are relatively large, few, and complicated, like human beings. Here prediction on a statistical basis is as important as ever, but, except for some aspects of psychology, the repeatable experiment is no longer at the centre of the study.... We then move into what are generally regarded as the humanities."[24]

Thus the student of, say, literature, faces a problem that does not trouble the chemist or the physicist or even the sociologist or the political scientist: while natural phenomena or perhaps even human behavior in the aggregate can be measured against the constant yardstick of random distribution, the examination of an artifact of the human will poses different perils. What does it mean to speak of a theoretical "random distribution" of a poet's words or phrases, in terms of which the significance of his choices or of his repetitions may be assessed? Tests of statistical validity generally require a random standard as a background, against which the "significance" of data becomes apparent by virtue of its difference from the random pattern. Any artifact of the human will, such as poetry, must therefore pose some philosophical difficulties for statistical analysis.

Conclusion

I will not attempt to offer even a rudimentary solution here,[25] but this problem serves as an example of the central problem created by the impingement upon the humanities of the scientist's world. Very likely, we must also steer clear of rigid "scientism," which "assumes a principle of determinism, i.e., from the quantitative description of an existing situation and from the known laws governing the changes of the magnitudes involved in the situation, a universal prediction can be made regarding the changes occurring in the situation in question.... Apart from measuring, observation and statistics, only the deductive method is considered to be valid."[26] Problems such as this one will not be resolved without recourse to the fundamental philosophy of interdisciplinary methodology. We have only begun to raise the right questions about how the quantitative analysis of literature that the computer makes more practicable is to be carried out.

References

1. The Two Cultures and the Scientific Revolution (New York: Cambridge University Press, 1959), p. 17. See also Snow, "The Two Cultures," New Statesman, October 6, 1956; F. R. Leavis and Michael Yudkin, Two Cultures? The Significance of C. P. Snow and an Essay on Sir Charles Snow's Rede Lecture (New York: Pantheon, 1963); Snow, The Two Cultures: and a Second Look (Cambridge: Cambridge University Press, 1964).

2. W. H. Auden, "Under Which Lyre: A Reactionary Tract for the Time (Phi Beta Kappa Poem, Harvard, 1946)," in Collected Poems, ed. Edward Mendelson (New York: Random House, 1976), p. 262.

3. "Biological Science and the Humanities: Some Considerations of Human Values and the New Biomedical Technologies," in Images and Innovations: Update '70's, ed. Malinda R. Maxfield (Spartanburg, S. C.: Center for the Humanities, Converse College, 1979), p. 123.

4. "The Humanities and Medical Science: An Epistemological Diagnosis," in Maxfield, ed., p. 137.

5. Hines, p. 127.

6. "The Future of the Humanities," National Forum: The Phi Kappa Phi Journal, 69, No. 3 (Summer 1979), 21.

7. "Reflections on the Humanities," National Forum: The Phi Kappa Phi Journal, 69, No. 3 (Summer 1979), 15.

8. "The Two Cognitive Dimensions of the Humanities," Daedalus, 99, No. 2 (Spring 1970), 256-57.

9. See for example Bruce A. Beatie, "Measurement and the Study of Literature," Computers and the Humanities, 13 (1979), 185; Beatie's article addresses similar problems from a perspective comparable to mine.

10. "The Humanities and Human Understanding," in The Humanities and the Understanding of Reality, ed. Thomas B. Stroup (Lexington: University of Kentucky Press, 1966), p. 13.

11. "Humanistic Studies: Their Object, Methods, and Meaning," Daedalus, 99, No. 2 (Spring, 1970), 247.

12. "Future," p. 23.

13. See The Making of Homeric Verse: The Collected Papers of Milman Parry, ed. Adam Parry (Oxford: Clarendon Press, 1971), and Lord, The Singer of Tales, (Harvard Studies in Comparative Literature, No. 24, Harvard University Press, 1960). For a general but dated account of subsequent work, see Edward R. Haymes, A Bibliography of Studies Relating to Parry's and Lord's Oral Theory, Publications of the Milman Parry Collection (Cambridge, Mass.: Center for the Study of Oral Literature, Harvard University, 1973).

14. Creditably, Joseph J. Duggan is unusually keenly aware of the statistical problems involved; but see his The Song of Roland: Formulaic Style and Poetic Craft (Berkeley: University of California Press, 1973), pp. 18ff.

15. Donald C. Green, "Formulas and Syntax in Old English Poetry: A Computer Study," Computers and the Humanities, 6 (1971), 92.

16. See, for example, Agnes M. Bruno, Towards a Quantitative Methodology for Stylistic Analysis (Berkeley: University of California Press, 1974).

17. Making Sense of Literature (Chicago: University of Chicago Press, 1977), pp. 26-27.

18. "The University and the Literary Public," in Stroup, ed., p. 69.

19. "Humanistic Studies," pp. 245-46.

20. "Speculation and Concern," in Stroup, ed., p. 36.

21. "Measurement," p. 186.

22. "The Aesthetic Implications of Formulaic Diction: How Are We to Read Beowulf?" unpublished paper read at the Conference on Language and Style, City University of New York, April 1977; see also Beatie, p. 189.

23. "La Chanson de Roland: An Oral Poem?" Olifant: A Publication of the Société Rencesvals, American-Canadian Branch, 4, No. 1 (October 1976), 63-74 (Beatie seems to have been unaware of this article); "Automatic Lemmatization in Serbo-Croatian," ALLC Journal (published by the Association for Literary and Linguistic Computing), 1, No. 2 (Autumn 1980), 55-59; "Statistics and the Computer in Formula Analysis of Serbo-Croatian Heroic Verse," Proceedings of the International Congress on Applied Systems Research and Cybernetics, ed. G. E. Lasker, 5, 2576-80 (Oxford: Pergamon Press, 1981); "Formulaic Style in Oral and Literate Opic Poetry" (with Lee Dunlap), Perspectives in Computing (published by IBM), 2, No. 4 (December 1982), 24-33. This research was aided by grants from the American Council of Learned Societies in 1979 and from the American Philosophical Society in 1980.

24. "Speculation," p. 37.

25. Though a comprehensive review of promising studies relevant to the problem at hand lies well beyond the scope of the present paper, the reader may find useful the bibliographies recently offered by Susan Hockey on pages 141-43 of A Guide to Computer Applications in the Humanities (Baltimore: Johns Hopkins, 1980) and by Robert Oakman on pages 170-71 and

217-227 of Computer Methods for Literary Research (Columbia: University of South Carolina Press, 1980).

26. The definition is that of Joseph Kockelmans, Phenomenology and Physical Science: An Introduction to the Philosophy of Physical Science, Duquesne Studies, Philosophical Series, 21, p. 75 (Pittsburgh: Duquesne University Press, 1966); for his account of the history of the trend of "scientism" in scientific thought, see pp. 72 ff.

A PERSONAL COMPUTER FOR EVERY COLLEGE STUDENT

David W. Bray

Educational Computing System
Clarkson College of Technology
Potsdam, NY 13676

Abstract

Clarkson College of Technology will provide each entering freshman student with a personal computer beginning in the fall of 1983. This paper describes the history behind the establishment of this program, how the program will operate and be administered, and some expectations as to how it may change the educational process at the college level.

Introduction

Clarkson College of Technology is one of the largest engineering colleges in the northeastern United States. It has concentrated in undergraduate education in engineering, science, and management for many years. Next fall, in August 1983, every freshman student entering Clarkson College will be provided with a powerful professional-quality Zenith Data Systems Z-100 desktop computer. By 1986 every undergraduate student, and most faculty and graduate students, will have his or her own personal computer. These computers will be interconnected via a campus wide network which will include the College mainframe computers as well as the personal computers. Clarkson is, to the best of our knowledge, the first college to provide every student with a personal computer and to integrate its use into the educational program.

The Zenith Data Systems Z-100 dual processor desktop computer, has 128K of program memory, high resolution bit-mapped graphics with 640 by 225 pixel resolution, one five and one-quarter inch floppy disk drive, and three communications ports. Each computer will be supplied with CP/M for use on the eight bit 8085 processor, Z-DOS (a derivative of MSDOS) for use on the sixteen bit 8086 processor, along with Fortran, Pascal, ZBasic (IBM PC compatible), Multiplan, and a word processor, as well as Cobol for those who need it. The computers will be used in all facets of the undergraduate educational programs, and will also be used by the faculty and graduate students for their research.

At the present time the College mainframe computers, an IBM 4341/Mod II and a Digital Equipment Corporation VAX-11/780, support over two hundred terminals. Even this amount of computation capability is not sufficient for the needs of the College. It has been a long standing policy to provide all students and faculty members with all of the computer services that they require. The requirements for service have been steadily growing, along with the waiting lines at the public terminals. It has been clear for some time that simply adding more and more terminals to mainframe computers is a losing battle. A solution for increasing computer resources is to put the new generation of powerful personal computers at the disposal of those who require it. At Clarkson this is the entire student body and faculty.

The introduction of the personal computer will do much more than provide the required computing power, it will change the educational process. Before we consider the ways in which we believe that education will change because of the computer, we next discuss how this program came into being, and how the Z-100 computer was selected.

Computer Selection

The introduction of the personal computer into the educational process at Clarkson had been under consideration for several years. Last year during the Spring 1982 semester a group of faculty and students formed a "class". They met on a regular basis twice weekly to consider the requirements and specifications of a personal computer and an associated campus wide network that would meet the College needs. The main recommendations of this group with regard to the personal computer were that:

1. The computer should contain a 16 bit processor. It was clear that the 16 bit processors were coming into the market in force, and thus it would not be wise to begin a new program with a computer operating on an eight bit processor, even though there was a large amount of software available for the eight bit processor computers. It was further felt that the computer should be compatible with the IBM Personal Computer (PC). The reason for this was that there are a large number of companies producing software for the IBM PC. There is also a fair amount of interest on the part of universities in the IBM PC. If this interest is strong enough, the IBM PC and compatible computers, might be the foundation for a standard in education. The benefits of a "standard" computer in education would be enormous, colleges could exchange software and courseware freely without the current problems

of computer incompatabilty.

2. The computer should have a high quality cathode-ray tube display that is capable of at least 80 characters per display line, and resolution high enough for engineering and scientific graphic applications. The 80 character display would allow the computer to provide professional word processing capabilities.

3. The computer should have a professional keyboard to complement the high quality display for word processing.

4. The computer should have bit-mappped graphic capabilities that would be suitable for engineering and scientific applications. If possible the graphics should be upgradeable to color, for those special applications that require it.

5. The computer should have communications ports that would be general enough to support network interconnection, and to support local printers should a student desire to attach one to his or her computer.

6. The computer should have software support that included at least: Fortran, Pascal, Basic, Cobol, word processing, and spread sheet accounting.

As a result of this study a set of guidelines were prepared. These guidelines were sent to the major personal computer manufactures which were believed to have, or be developing, computers which might meet our needs. These guidelines suggested that the program might be implemented as early as the fall of 1984. A wide variety of responses were obtained. The first manufacturer to respond indicated that they had a computer "on the drawing board" that exceeded our specifications, and suggested that our timetable could be pushed up to the fall of 1983. Being encouraged by this response we decided to attempt implemention of the plan by the fall of 1983.

Early in the summer of 1982 we prepared a formal set of specifications for the computer, and sent it to those who received our original guidelines, and others who had heard of our project. From this effort we received bids from five companies. From these the Zenith Data System Z-100 was selected for its price/performance combination. The Z-100 computer exceeded our specifications in almost all ways. It has two features that were considered most important. The first of these is a second processor, an 8085. This will allow the current eight bit software to be used on the computer if desired. This additional processor removes one of the concerns that a 16 bit computer would limit use of available software. The second feature is the inclusion of four available S-100 industry standard bus slots. This allows expansion of the computer for use in the laboratory. The fact that the student computer would be software compatible with the laboratory computers would be very beneficial.

Implementation Plan

At Clarkson the plan for the introduction of the personal computer is to start with the freshman class entering in the fall of 1983, and to continue with each entering freshman class thereafter. Ownership of the personal computers will reside with the College. However, when the student graduates the ownership will be transferred to the student. Should the student withdraw from the College without completing degree requirements, the computer will remain with the College. These computers will be re-assigned to transfer students. Beginning with the entering freshman class, there will be a tuition surcharge of $200 per semester to cover, in part, the additional cost of the program. In addition, each student will be required to make a $200 maintenance deposit to be used in the event that servicing of the computer is necessary. It is the students responsibility to keep the computer in good condition. After the ninety day guarantee period has expired the student will be financially responsible for keeping it maintained. The College will insure the computers against fire and theft.

This implementation plan has a number of benefits. First, having the personal computer introduced over a four year period with each new entering class provides lead time for the faculty of the upper class courses to modify and develop their courses for the most effective use of the computer. Second, the graduating students will take with them a computer and much self-developed software that will help bridge the gap from college to business and industry. Third, the computer can be a potential means of communications and contact with alumni. It might even be a vehicle for continuing education. Fourth, the College will be free to select new model computers as technology advances since each year new computers will be needed for the entering freshman class. Thus the computing facilities will not grow old in time as do the current facilities involving mainframe computers.

Following this plan, this fall each freshman will be provided with a computer. For many years every freshman student has been required to take a computer programming course. With the introduction of the personal computer the subject matter of this courses will not change substantially. What will change is the availability of a computer to the student. The course has been modified to take advantage of ready access to a computer and, of course, to teach the students how to use the particular features of the Z-100 computer to full advantage. Most students will not have had much computer programming training prior to entering Clarkson, and almost none will be proficient with the Z-100, its operating systems, and language processors. Therefore, the other first term freshman courses must restrict their use of the computer to those problems that do not require extensive programming knowledge, or to demonstration programs that are supplied to the student in a pre-programmed form. In the second semester freshman year, the students will be able to do

more programming on their own. Many will have become very proficient with their computers. As they progress from semester to semester the professors will be able to require more programming knowledge.

To allow the faculty time to discover the many potential uses of the personal computer in their courses, computers have already been provided to a number of faculty members. Those who first obtained the computers have contact with freshman and sophomore students (or other students who will be using the Z-100 in the laboratories), and wish to introduce the use of the computer into their courses. The initial delivery of computers for the faculty was completed before the end of the Fall 1982 semester. This gave a minimum of nine months lead time for course preparation planning for the freshman courses, and more than that on the average. Those faculty members who are teaching upper divison courses will be receiving computers in the near future.

Administration of the Program

To administer the introduction of the personal computer into the educational process at Clarkson, a new position has been established. This position is the Dean of the Educational Computing System. The responsibilities of this position are, in general terms:

1. To provide coordination for academic activities concerned with the program.
2. To develop a personal computer network which will interconnect with the current mainframe network that now exists on campus.
3. To administer the purchase, distribution, and repair of the personal computers.

A major responsibility will be in coordinating the academic activities. This will include training of the faculty in the use of the Z-100 operating systems and associated software, and coordinating course development in the sense of disseminating information about the activities of the faculty and the software that is being developed. It will also include conducting workshops at the local college level so that faculty members may learn from fellow faculty members who have introduced the Z-100 in their courses, and holding workshops led by professors from other colleges who have used personal computers in various academic settings.

At the present time, the college mainframe computers and several groups of remote terminals are tied into a X.25 packet switching network. The present thinking is that the personal computers will be tied into this network by forming a series of local area networks consisting of several hundred personal computers each. Each local networks will be tied into a X.25 packet switcher and thereby connected to the main network. This network is under study at this time. It is expected that it can be operational with the personal computers within two years.

Repair of the computers, after the ninety day warranty period, will be conducted by college personnel. Being a technical college there is no lack of qualified technicians and students already on campus.

Educational Benefits

The computer has become a necessary tool in the business world. After graduation current students will be confronted with computers in their employment, no matter what the field. A tool that has become so much a part of business and industry, surely has an equally important place in the educational process. At this point in time it is very difficult to even imagine the effect that the personal computer will have on the educational process. However, there are a few clues as to what may occur. Just as the calculator replaced the slide-rule in engineering and science courses, the computer will replace the calculator. Its potential in education, however, is much more than simply being a better tool to do the same job.

Some of the important advantages of the computer in education are evident now. These are:

1. When a student is asked to perform a homework assignment that involves a large amount of calculation the process of performing the calculations becomes the focus of the lesson. That is, the student studies the theory, organizes the problem to be solved, and then begins the calculation. At that point the process of operating the calculator receives the full attention of the student. The student often loses sight of the lesson, concentrating only on pushing the calculator buttons. If the student is asked to perform the same problem on a computer with which he is familiar, the solution takes a different form. The student organizes the theory in terms of language of the computer. A language much like the equations of the theory. Therefore the student's efforts toward obtaining the calculation actually becomes an effort of rewriting and rethinking the theory of the problem. The computer can then focus its attention on obtaining the actual solution. A student who can instruct a computer how to solve a problem has a good grasp of the problem.
2. Computer simulations can be useful and effective teaching tools. The computer can aid the student in obtaining more insight into scientific principles that are being studied. Once a student has obtained a solution to a problem that has been calculated by a computer, there is very little effort involved in obtaining more solutions to the same problem with different input parameters. The ability to obtain many solutions to a problem in a matter of a few minutes can provide the student with intuition about the subject matter that is generally not possible without a computer. Particularly, if a solution to a problem is presented in a form that is easily understood, such as graphically, a student can learn much

about a subject matter by changing parameters and seeing the effect upon the result.

3. The computer can in many cases be a substitute for laboratory experiments at the introductory level. For economic reasons many colleges have had to eliminate introductory chemistry and physics laboratories. The personal computer can provide, in simulated form, much of the information that is gained in the laboratories. It, of course, cannot substitute for the experience gained in working with the physical objects of an actual laboratory.

   In those subjects which deal with complex systems such as the stock market, management systems, weather patterns, etc. the computer can, through simulation, give the student valuable albeit limited experience that could only otherwise be obtained in the real world.

4. The computer has the potential for demonstrating principles that are difficult to present in lecture form. Lectures can be supplemented by the student executing programs prepared by the course instructors which would demonstrate, possibly in graphical form, the fundamentals of the lecture subject matter. In a sense, this is another form whereby the student can gain more intuition into the subject matter at hand.

Without question the personal computer will have a dramatic effect on the educational process. Only after a few years of experience at the college level will we be able to access the full impact that the computer can have in the educational process. Clarkson College is committed to provide leadership in the use of the personal computer in college education.

COMPUTER-ASSISTED SIMULATION IN THE POLITICS OF REAPPORTIONMENT/REDISTRICTING (CASPOR)

Jerry E. Bolick and James O. Icenhour

Lenoir-Rhyne College, Hickory, North Carolina 28601

ABSTRACT

Written in Basic and implemented on a PDP-11/34 system, CASPOR is designed to enhance the learning process in both computer science and political science classrooms. It provides a realistic project for computer science students and renders manageable the classroom simulation of the politics of reapportionment and redistricting by political science students. CASPOR is flexible, can be adapted to various classroom situations, and can be modified to reflect the political realities of various states.

INTRODUCTION

The program presented and described herein is called Computer-Assisted Simulation in the Politics of Reapportionment/Redistricting (CASPOR). This program grew out of the desire of one co-author for improved instructional methods in the teaching of the political dimensions of the reapportionment/redistricting phenomenon at the state level and the need of the other co-author for more realistic projects for his students in a data structures course. CASPOR has been tested and used in our classrooms with excellent results. We believe this program is sufficiently flexible to permit its adaptation to the political realities of various state political systems and that it can be readily modified to fit the classroom needs of a wide range of classes as well.

CASPOR is written in the programming language Basic and is implemented on a PDP-11/34 computer system. Data files, all of which are stored in virtual arrays, are used extensively. The structure of these files is an essential ingredient in this simulation.

The data for this simulation were derived from the public records of North Carolina. Variables include population, voter registration by political party preference, and presidential, congressional and gubernatorial election results, all by county.

This paper describes the step by step process by which computer science students, working in teams, amassed the data base from original sources, selected appropriate structures for the data to maximize the efficiency of the program and minimize the use of available computer memory, developed the program, and tested its various components.

The application of this simulation in the political science classroom involves the use of an outline map of North Carolina with county boundaries detailed, student role assignments, general rules and limitations, and the use of CASPOR for analyzing and verifying proposed redistricting plans.

DEVELOPING THE PROGRAM

The program CASPOR was developed with the aid of a beginning class in data structures. After the class had analyzed the problems of reapportionment/redistricting and understood exactly what was necessary to produce a useful program, the class of 12 students was divided into 4 groups of 3 each. Under the professor's direction and coordination, the groups completed the following assignments:

1. Collected voting, registration and population data for each county and stored it in a data file.
2. Wrote a subroutine to determine if the counties in a proposed district are contiguous.
3. Wrote a subroutine that permits modification of districts.
4. Wrote a subroutine to print the information about each district and calculate cumulative statistics for each district.

The simulation makes extensive use of data that must be available each time the program is used. One of the essential elements of the simulation is the 23 items of data which describe the population, registration and voting statistics for each of the 100 counties in North Carolina.

The structure selected for this data is a 100 x 23 array - one row for each of the 100 counties and one column for each of the 23 data items for each county. Since this data set must be available to the program at all times and is quite large for a PDP-11/34 computer, the students chose to store the data in virtual array.

When the political science students begin their reapportionment/redistricting simulation, they can use the computer as their data source.

To collect the needed data the student runs CASPOR and selects the data collecting option.

When the political science students collect the data needed for their decisions, they are also supplied with a list of North Carolina's 100 counties which are listed in alphabetical order and indexed with numbers 1 - 100. The counties are stored in an array that contains 100 elements so that each county can be referenced by its index number. Again, the data structure class chose a virtual array so that the list of counties can be stored separately from the main program and yet can be accessed by the program when proposed districts are being developed.

With the statistical data related to population, voter registration and election results accessible, the next problem that must be solved is that of contiguousness. Since all counties in a congressional district must be contiguous, the computer must be able to decide if the counties selected for a district are in fact contiguous.

The data structures class found that the problem of county contiguousness is really an application of graph theory. If one lets the nodes of a graph represent counties and agrees that two nodes are joined by an edge if two counties are adjacent, a graph that describes the adjacency of counties in North Carolina results. Figure 1 shows a segment of this graph.

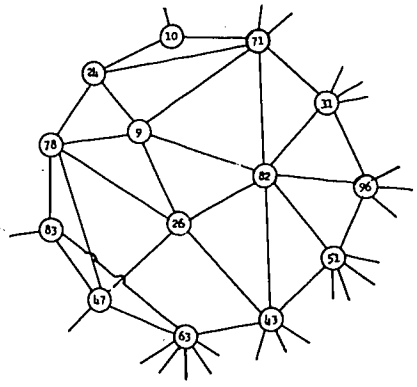


Figure 1

An obvious way to represent this graph is to use a 100 x 100 adjacency matrix

$$A = [a_{ij}] = \begin{cases} 1; & \text{if counties i and j are adjacent} \\ 0; & \text{otherwise} \end{cases}$$

But this representation produces a sparce matrix and is an inefficient use of memory. Hence, the class chose a matrix representation in which the entries in row i are the indices of the counties that are adjacent to county i. For example, Bladen County has index number 9 and counties with indices 24, 26, 71, 78, and 82 are adjacent. Thus Row 9 of the adjacency matrix has 24, 26, 71, 78, and 82 as its non-zero entries (a zero indicates no more counties in the adjacency list). This matrix is also stored as a virtual array. The following portion of the matrix describes the adjacency of North Carolina counties.

```
...
 9  78  26  82  71  24   0   0   0   0   0
...
24  78   9  71  10   0   0   0   0   0   0
...
26  67  47  63  43  82   9   0   0   0   0
...
71  65  10  24   9  82  31  67   0   0   0
...
78  83  47  26   9  24   0   0   0   0   0
...
82  71   9  26  43  51  96  31   0   0   0
```

Using this matrix, it is relatively easy to write a subroutine that searches for a path through all of the counties that are proposed for a district. If a path exists then the counties are contiguous. The following program segment searches for a path through a proposed district.

```
4105 S1(L1)=0 FOR L1=1 TO 90
4120 OPEN "ADJNCY.DAT" AS FILE #8
4200 S=D1(1)
4210 FOR C=1 TO 20
4220 IF D1(C)<>0 THEN 4240
4225 CLOSE #8
4230 F$="YES"
4235 RETURN
4240 T=D1(C)
4250 IF T=S THEN 4450
4260 L=1
4263 S1(L)=S
4270 I1=2
4280 I2=S1(L)
4290 IF S1(L)<> 0 THEN 4310 ELSE F$="NO"
4295 CLOSE #8
4300 RETURN
4310 J5=0
4320 J5=J5+1
4330 IF A4(I2,J5)=T THEN 4450
4335 IF A4(I2,J5)<>0 THEN 4340
4337 L=L+1\GOTO 4280
4340 FOR K=1 TO 20
4350 IF A4(I2,J5)<>D1(K) THEN 4400
4360 FOR M=1 TO I1-1
4370 IF A4(I2,J5)=S1(M) THEN 4400
4375 NEXT M
4380 S1(I1)=A4(I2,J5)
4385 I1=I1+1
4390 GOTO 4320
4400 NEXT K
4410 GOTO 4320
4450 NEXT C
```

If a group of counties constitutes a valid district, it is then necessary to determine the cumulative population, election results and voter registration statistics for the district. In particular, it is essential that the population of each district be recorded since the population of each district must be compared to the average population per district in terms of deviation and average deviation. These deviations must fall within predetermined bounds.

If these bounds are not satisfied then the

district must be adjusted. Therefore, the data structures class wrote a subroutine to permit counties to be moved from one district to another.

North Carolina actually has only 11 Congressional Districts, but for this simulation 12 districts are used with a maximum of 20 counties per district. The indices of the counties that are selected for district i are stored in row i of a 12 x 20 matrix, C. After composition of the 12 districts has been determined, the print subroutine uses matrix C to sort the population, registration, and voting data for the counties in a district, calculates the totals for the district, and prints a complete description of the district. The district totals are saved and the final printout is a set of cumulative statistics for each of the districts. Appendix 1 contains a complete description of the proposed 5th District.

The summary analysis in Figure 2 shows total population, deviation in population, and relative deviation in percent for each proposed district.

TOTAL STATE POPULATION	5,084,360
RATIO OF HIGHEST TO LOWEST	1.338
NUMBER OF CONGRESSMEN	12
AVERAGE POP PER CONGRESSMAN	423,696

DIST	POP	DEVIATION	RELATIVE DEVIATION
1	354,040	-69,656	-16.4402
2	445,257	21,561	5.0887
3	473,794	50,098	11.8240
4	419,394	-4,302	-1.0154
5	415,252	-8,444	-1.9930
6	457,354	33,658	7.9438
7	405,817	-17,879	-4.2199
8	428,202	4,506	1.0634
9	429,285	5,589	1.3190
10	410,409	-13,287	-3.1361
11	420,830	-2,866	-0.6765
12	424,722	1,026	0.2421

AVERAGE RELATIVE DEVIATION PER CONGRESSMAN	4.5802
AVERAGE DEVIATION PER CONGRESSMAN	19,406

Figure 2

IMPLEMENTING THE PROGRAM

The implementation of the simulation (CASPOR) in the political science classroom begins with the assignment of roles to the various participants. Collectively the class constitutes a simulated state legislative committee charged with the task of devising a congressional redistricting plan to accomodate North Carolina's presumed increased apportionment of congressional seats. Each student is assigned a role that includes specification of the following variables: county of residence, political party affiliation, age, sex, race, religion, social status, economic status, and legislative seniority and assignment (Appendix 2).

All of this information is public knowledge and is shared with all other participants (Appendix 3). In addition to this public role, each student is provided with a statement of confidential political considerations which is not shared. Each participant is urged to maintain this assigned role in utmost confidence (Appendix 4).

The overall rules of the simulation are then explained to the participants. Each is urged to maintain high fidelity to the assigned role in all negotiations among the group. Specific limitations include the following:

1. Districts must be as compact as possible.
2. Districts must include whole counties.
3. Exact mathematical equality of district population is desired.
4. Overt gerrymandering will render proposed districting plans suspect and should therefore be consciously avoided or minimized.

An outline map of North Carolina is provided each participant and is to be used to outline proposed district boundaries (Appendix 5). The participants are now directed to the computer and given access to all the data stored in the CASPOR program.

After considerable individual efforts to develop acceptable district plans, the participants convene as a group and begin the process of negotiations necessary to reach workable and acceptable solutions to the redistricting problem. When a consensus is reached it is tested and verified by the application of the CASPOR program. Remodification and revalidation are pursued until either an acceptable redistricting plan is devised or until stalemate is reached.

When the simulation is terminated the participants then share their confidential roles with each other and the instructor summarizes the learning experience mutually shared by the participants in the simulation.

SUMMARY

While CASPOR was developed for use in both computer science and political science courses, it can be used independently in either. Furthermore, any number of students can participate. Small classes can participate as a whole while large classes can be sub-divided into smaller teams with each team developing or using the complete simulation. Roles can be assigned to reflect any constellation of political variables. And finally, the concept can be applied to the political systems of other states.

CASPOR has enhanced the learning process in our classrooms. We hope that others may find it useful in their classrooms as well.

Appendix 1

DISTRICT # 5

POPULATION STATISTICS

COUNTY	TOT POP	TOT WHITE	% WHITE	TOT BLACK	% BLACK	TOT OTHER	% OTHER
BLADEN	26,477	16,151	61.00	10,326	39.00	0	0.00
COLUMBUS	46,937	32,997	70.30	13,940	29.70	0	0.00
CUMBERLAND	212,042	161,364	76.10	50,678	23.90	0	0.00
ROBESON	84,842	62,953	74.20	21,889	25.80	0	0.00
SAMPSON	44,954	29,445	65.50	15,509	34.50	0	0.00
TOTALS	415,252	302,909	72.95	112,343	27.05	0	0.00

REGISTRATION STATISTICS

COUNTY	TOT REG	TOT DEM.	% DEM.	TOT GOP.	% GOP.	TOT OTHER	% OTHER
BLADEN	13,638	12,629	92.60	857	6.28	152	1.11
COLUMBUS	24,831	22,377	90.12	2,147	8.65	307	1.24
CUMBERLAND	57,936	44,536	76.87	8,938	15.43	4,462	7.70
ROBESON	48,340	45,300	93.71	2,357	4.88	683	1.41
SAMPSON	23,734	14,571	61.39	8,621	36.32	542	2.28
TOTALS	168,479	139,413	82.75	22,920	13.60	6,146	3.65

PRESIDENTIAL ELECTION STATISTICS

COUNTY	TOT VOTE	TOT DEM.	% DEM.	TOT GOP.	% GOP.	TOT OTHER	% OTHER
BLADEN	7,589	6,009	79.18	1,546	20.37	34	0.45
COLUMBUS	14,401	11,148	77.41	3,184	22.11	69	0.48
CUMBERLAND	38,683	24,297	62.81	14,226	36.78	160	0.41
ROBESON	25,699	20,695	80.53	4,907	19.09	97	0.38
SAMPSON	15,902	8,869	55.77	6,968	43.82	65	0.41
TOTALS	102,274	71,018	69.44	30,831	30.15	425	0.42

SENATE ELECTION STATISTICS

COUNTY	TOT VOTE	TOT DEM.	% DEM.	TOT GOP.	% GOP.	TOT OTHER	% OTHER
BLADEN	5,140	3,093	60.18	2,047	39.82	0	0.00
COLUMBUS	9,630	5,610	58.26	4,020	41.74	0	0.00
CUMBERLAND	25,345	12,358	48.76	12,987	51.24	0	0.00
ROBESON	12,156	7,296	60.02	4,860	39.98	0	0.00
SAMPSON	14,609	6,423	43.97	8,186	56.03	0	0.00
TOTALS	66,880	34,780	52.00	32,100	48.00	0	0.00

HOUSE ELECTION STATISTICS

COUNTY	TOT VOTE	TOT DEM.	% DEM.	TOT GOP.	% GOP.	TOT OTHER	% OTHER
BLADEN	6,665	5,853	87.82	812	12.18	0	0.00
COLUMBUS	14,466	12,904	89.20	1,562	10.80	0	0.00
CUMBERLAND	37,591	30,125	80.14	7,466	19.86	0	0.00
ROBESON	25,613	23,103	90.20	2,510	9.80	0	0.00
SAMPSON	15,966	8,654	54.20	7,312	45.80	0	0.00
TOTALS	100,301	80,639	80.40	19,662	19.60	0	0.00

GOVERNOR'S ELECTION STATISTICS

COUNTY	TOT VOTE	TOT DEM.	% DEM.	TOT GOP.	% GOP.	TOT OTHER	% OTHER
BLADEN	7,363	6,432	87.36	861	11.69	70	0.95
COLUMBUS	14,216	11,994	84.37	2,152	15.14	70	0.49
CUMBERLAND	39,020	28,646	73.41	9,654	24.74	720	1.85
ROBESON	24,908	22,212	89.18	2,539	10.19	157	0.63
SAMPSON	15,810	9,718	61.47	5,980	37.82	112	0.71
TOTALS	101,317	79,002	77.98	21,186	20.91	1,129	1.11

Appendix 2

CASPOR ROLE ASSIGNMENT

STUDENT'S NAME: _____ AGE: _____

REPRESENTATIVE FROM: _____ (County) POLITICAL PARTY AFFILIATION: _____

SEX: _____FEMALE _____MALE RACE: _____WHITE _____BLACK _____OTHER (SPECIFY)

RELIGION: _____MAINSTREAM PROTESTANT _____FUNDAMENTAL PROTESTANT _____CATHOLIC _____JEWISH

ECONOMIC STATUS: _____UPPER _____UPPER MIDDLE _____MIDDLE _____LOWER MIDDLE

SOCIAL STATUS: _____UPPER _____UPPER MIDDLE _____MIDDLE _____LOWER MIDDLE

LEGISLATIVE SERVICE (SENIORITY): _____

_____

****

CONFIDENTIAL POLITICAL CONSIDERATIONS
(Must Not Be Shared With Other Participants)

_____

_____

Appendix 3

SUMMARY OF CASPOR ROLE ASSIGNMENTS

STUDENT'S NAME	COUNTY	PARTY	AGE	SEX	RACE	RELIGION	ECONOMIC STATUS	SOCIAL STATUS	LEGISLATIVE SENIORITY	COMMITTEE ASSIGNMENTS
Able	Mecklenburg	R	63	F	W	J	U	U	6	V Ch House Comm on Humanities and Fine Arts
Baker	Madison	D	55	M	W	FP	UM	M	16	Ch House Comm on Appropriations
Charlie	Durham	D	45	M	B	MP	UM	UM	8	Ch House Comm on Finance (Taxes)
Delta	Guilford	D	40	F	B	MP	M	M	4	House Comm on Social & Economic Affairs
Easy	New Hanover	D	52	M	W	C	U	UM	14	Ch House Comm on Natural Resources
Foxtrot	Stanly	R	48	M	W	MP	M	M	4	House Comm on Commerce
George	Forsyth	D	40	F	W	C	U	U	6	Ch House Comm on Labor
How	Cumberland	D	35	M	B	MP	M	M	4	V Ch House Comm on Transportation
India	Surry	R	30	M	W	FP	LM	M	-	1st term
James	Wilson	D	60	M	W	MP	U	UM	12	Ch House Comm on Agriculture
King	Buncombe	R	48	M	W	C	UM	UM	8	Leader House Minority
Love	Wake	D	56	F	W	MP	UM	UM	6	Ch House Comm on Justice

Appendix 4

SUMMARY OF CASPOR PARTICIPANT'S
CONFIDENTIAL POLITICAL CONSIDERATIONS

Able. Needs to retain seat for enhancement of personal social status. Political party is of secondary importance. Passionately desires to see personal friend (also Republican) capture seat in U. S. Congress (from Mecklenburg County) next election.

Baker. Wants to become first black U. S. Congressman from North Carolina in modern history. To do this, Durham County must be part of a congressional district that encompasses northeastern counties with large black populations. Everything else is of secondary importance. But does have intense loyalty to own race and political party.

Charlie. Wants to become Speaker of the House, so must keep own seat and curry favor with all other legislative Democrats as well as Democratic party leaders statewide. Seeks good publicity and party harmony.

Delta. Wants to see U. S. congressional districts drawn in a way that will make the major metropolitan centers of North Carolina the focal points of congressional districts, thus enhancing the over-all political impact of black voting concentrations.

Easy. Over-riding desire is to see the entire outer-banks area of the state encompassed in a single U. S. congressional district. Or at least as much of it as can be achieved. Former N. C. State Democratic Party Chairman with vast real estate holdings in the outer banks area. Would probably run for Congress from such a district.

Foxtrot. Views Stanley County as part of the eastern section of the state. Wants to make sure Stanly County is not placed in a congressional district with any counties to the west of it. Would also like to see district lines drawn to enhance Republican voting power in congressional elections.

George. Passionately anti-organized labor. Wants Forsyth County to be included with rural anti-labor counties in a congressional district so as to elect to U. S. Congress a senior vice president of one of the Winston-Salem tobacco firms. Is loyal to Democratic Party and rather liberal.

How. Owns local independent trucking firm. Served in U. S. Army as officer for 4 years. Relates well to military personnel at Ft. Bragg and Pope AFB. Wants to be a king-maker by providing winning margin in next election for new congressman from Fayetteville. Hopes for appointment as U. S. Assistant Secretary of Transportation.

India. Wants to be Lt. Governor, or U. S. Congressman by age 40. Politically ambitious but adheres to a very strict personal code. Will sacrifice principle only for personal political advantage or gain.

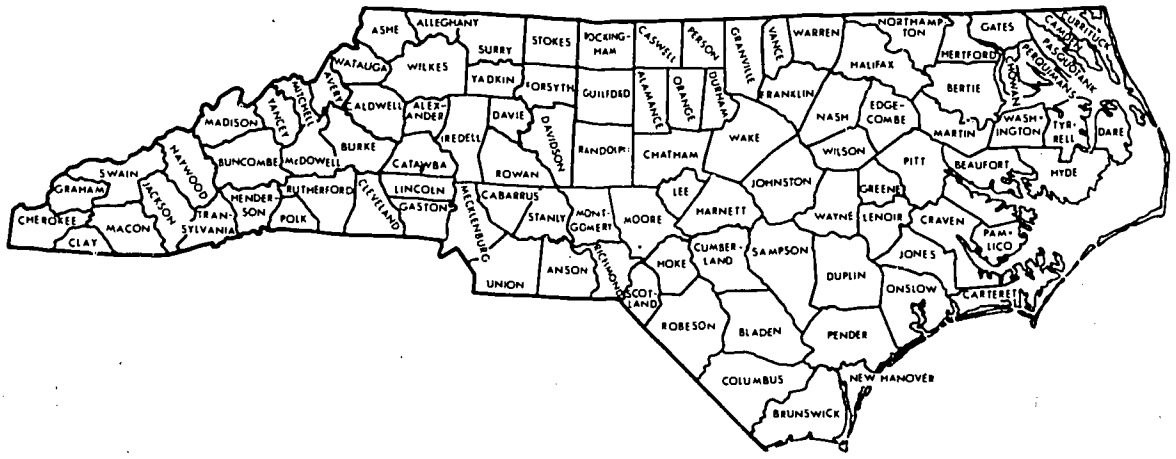
James. Tobacco farmer and warehouseman. Dedicated to his own economic self-interest. Above all else, wants to keep high party power position and ability to horse-trade in the legislative process.

King. Hopes to win seat in U. S. Congress from a western North Carolina congressional district. Must structure a congressional district that includes Asheville in a way that will enhance Republican chances of winning next election. Strong party supporter and one who thrives on publicity.

Love. Wants to run for Office of N. C. Attorney General or N. C. Supreme Court. Lawyer of some note, with thriving practice in Raleigh. Seeks greatest exposure statewide to create positive personal image. Seeks party unity and would like to see all N. C. congressional districts elect Democrats.

Appendix 5

NORTH CAROLINA



INTEGRATING COMPUTING PACKAGES AND STATISTICS INSTRUCTION

William D. Schafer and C. Mitchell Dayton

Department of Measurement, Statistics, and Evaluation
College of Education, University of Maryland
College Park, Maryland 20742

Abstract

A computer-based, intermediate-level applied statistics course is described. All course assignments were carried out using BMDP and SPSS routines and the results were integrated into the course instruction. An evaluation of the course is discussed and revisions based on the evaluation are presented. A description of the data base and the assignments are included.

Introduction

Many statistics courses have as one of their goals providing a computational basis with which statistical topics can be applied in practical situations. This goal is often met using devices such as calculators which have undergone recent changes in power and portability, but have retained limitations in terms of storage capacity and output. With good reason, field applications of statistics primarily use computers and there exists a large body of software to support this use. It is natural, therefore, that applied statistics curricula should increasingly reflect involvement with computers.

Computing packages in institutions which are users of statistics have become commonplace. While they are somewhat restrictive in that one is limited to the analyses they provide, they do employ a rather broad coverage of popular techniques and are relatively easy to use.

A course is described here which was developed in order to provide package programming skills to students in support of the topics often found in a second-semester applied statistics curriculum. In this effort we had three guides: the traditional content of such a course, the capabilities of the local computing center, and an article by Thisted[3] describing similar efforts.

The course was implemented first during summer of 1979. This paper describes an evaluation of the course during the fall of 1979, when a non-computer-oriented section was available as a "control". Finally, revisions to the course based on these experiences are suggested and an updated version of the course is discussed.

Characteristics of the Course

Materials. It was expected that students would have had experience with some computing package in a first course in applied statistics. Materials used in a first course administered locally were used when students did not have this background.

The textbook (Afifi and Azen)[1] was supplemented with various handouts and descriptions of BMDP and SPSS packages (those used in the course) developed locally. Use of the full manuals was encouraged but not required. Single copies of these materials and the syllabus may be requested from the first author.

Topics and Assignments. Following an initial introduction to BMDP and SPSS, the order of topics was: bivariate regression and correlation; part and partial correlation; multiple regression and correlation; dummy coding and product variables; ordered regression; one-way analysis of variance; comparisons and contrasts; simultaneous inference; two-way analysis of variance; interpretation of interaction; non-orthogonal analysis of variance; fixed, random, and mixed models; analysis of covariance; generalized ordered regression; Hotelling tests; discriminant analysis; multivariate analysis of variance; and principal components analysis. The topics were illustrated throughout the semester using a total of 36 final computer runs by each student, equally split between BMDP and SPSS programs.

The data set for the majority of assignments was taken from the 1979 Information Please Almanac and consisted of sixteen variables relating to the "quality of life" of each of 113 nations, along with their continents. Arbitrary codes were used for missing data. A description of the data set and the assignments appear in the appendix.

Students were assigned corresponding computer runs for virtually every topic which was presented. By-hand computations were included in most cases, as well. In three instances data from the text were used to give students experience in preparing data for processing by computers.

Students. Forty-two students completed the course. Of these, thirty-eight (90%) were pursuing a doctorate. Their programs were in the areas of human development (12, 29%), counseling and personnel services (11, 26%), secondary education (6, 14%), health education (3, 7%), industrial education (3, 7%), special education (1, 2%), and measurement and statistics (6, 14%), the home department of the course. Twelve students dropped during the semester, representing 22% of the original registration, which is not particularly unusual for this course.

Format. The course was taught on a twice-perweek basis over one semester (16 weeks) by a single instructor with no graduate assistant support. Computer facilities were located some distance from the classroom. Most students operated in a batch mode because of inaccessibility of terminals and file space on the campus. Batch turnaround varied between five minutes and six hours during the semester.

Topics were treated for the most part with an introductory presentation of the material followed by a discussion of package output in a later class period, thus allowing students to refer to their own completed assignments (and annotate them) during the session. Assignments were collected and returned at the next class.

Student Evaluation

Procedures. Two sections of the course were offered during the fall semester of 1979, one on two mornings (9:30 - 10:45) and one on two afternoons (4:15 - 5:30) each week. The section given on the afternoons was instructed using the computer-oriented approach described here; the other section was instructed by another faculty member of the same rank using a non-computer-oriented approach traditionally used in the department. Aside from incidental information learned during the semester, the students were unaware of the difference in approaches. With minor exceptions, identical content was covered. The syllabus for the "control" section may be requested from the first author.

Students were asked to complete an attitudinal inventory covering their expectations in the course during the initial class period. An anonymous method was used to match these "pretests" with "posttests" covering identical content at the end of the course. also, students completed a ten-item pretest covering prerequisite material during the first class session. These were matched with their performance on forty-eight common items embedded in the two examinations in each of the sections.

Subjects. Students registered for the two sections normally. The "control" section originally registered 35 students of whom 19 (54%) completed the course. The "computer" section originally registered 54 students of whom 42 (78%) completed the course. The difference between the proportions is significant (corrected $x^2 = 4.40$).

Results. Scores on the common exam items were paired with the corresponding item on the pretest. The observed difference between the sections was not significant (see table 1) at conventional levels.

The ten items on the affective posttest were each paired with the corresponding item on the pretest. The observed differences on these items were significant at conventional levels in favor of the "computer" group (see table 1):

I have come to understand many useful statistical techniques in the course.

I learned many new skills which will benefit me in my research in the future.

I had difficulty understanding the material presented in class (reversed scale).

This course was particularly well-organized.

This course increased my interest in statistics.

I enjoyed my experience in this course.

The observed difference on this item was significant at conventional levels in favor of the "control" group:

I had difficulty understanding the material presented in the textbook (reversed scale).

The observed differences on these items were not significant at conventional levels:

I spent too much time and effort on this course (reversed scale).

I had adequate background preparation to take this course

The material in this course was covered too superficially (reversed scale).

Observations

Thisted[3] presented three organizational approaches for such a course: by package, by topics, and by software systems components. We chose to organize by topics. This method has two major advantages: it maintains a logical flow of content, building later topics upon earlier material, and it allows comparisons to be made between the computer packages on features relevant to potential statistical analyses.

The data set has some advantages but we feel it could be improved upon. One clear advantage is its real, public nature (it can be located and understood easily). Also, it contains some categorical variables, exhibits some interesting relationships, and presents some common difficulties such as missing data and outliers, some which arise through obvious error in the data set. However, the data set is not educationally

oriented. It would be valuable to have a set which is more meaningful while maintaining the desirable elements noted.

We have found that it is very important to include some "by-hand" work based on each output. If this is not done, there is a tendency for some students to regard their output as a final product with no investigation of its meaningfulness until class time. Also, since it is important that all students have the same output for discussion purposes, we have found it helpful to give students some significant numerical value to look for in the output in order to check their work prior to class.

As noted in the evaluation, textbook support proved to be a problem. We have had success recently with Pedhazur[2].

Our experience with this course has been positive. Students seem to develop facility with BMDP and SPSS in this context and seem to appreciate the tools they have learned how to use in support of what they have learned about statistics. While the design of the course evaluation is admittedly modest, the results do not dissuade us from this view.

References

1. Afifi, A.A. and Azen, S.P. Statistical Analysis: A Computer Oriented Approach. Academic Press, 1979, 2nd ed.

2. Pedhazur, E.J. Multiple Regression in Behavioral Research. Holt, Rinehart, and Winston, 1982, 2nd ed.

3. Thisted, R.A. "Teaching Statistical Computing Using Computer Packages." The American Statistician, 1979, 33(1), 27-30.

Computer Assignment No. 1
Due on Class Meeting Number 4

A. The element EDMS*STAT.QUALITY contains data for 113 nations which relate to the "Quality of Life" in those nations. A description of the variables, the data format, and a source for the data are presented in the element EDMS*STAT.QUALFORMAT. A listing of that element is to be obtained by setting up a run including the following statement:

@PRT,S EDMS*STAT.QUALFORMAT

This data set will be used in this and several subsequent assignments. To include the data in an SPSS or BMDP run, use

@ADD EDMS*STAT.QUALITY

when the data are needed (e.g., after a READ INPUT DATA card in an SPSS run or after the /END card in a BMDP run).

B. For the variables POP and LITER, generate histograms, ogives, and normal probability plots using BMDP5D. Similarly, generate a histogram for the variable FREEDOM using SPSS FREQUENCIES.

C. Use SPSS CONDESCRIPTIVE to obtain summary statistics for all 17 variables; also, use BMDP1D for the same purpose.

D. From summary statistics on the printouts in C., above, use a one-sample t test to test the hypothesis that the true value for mean energy consumption is 2000.0 kg of coal equivalent. Also test the hypothesis that the true variance for this variable is 5 million square units. These computations must be done by "hand."

E. Using two-sample t tests, compare Asia and Africa on all of the remaining 16 variables. Run the analyses with both SPSS T-TEST and BMDP3D. Show in detail (by "hand") how the t test for the variable GNP is computed.

F. Prepare a bivariate plot for the variables BIRTHS (vertical axis) and LITER (horizontal axis). Use SPSS SCATTERGRAM and BMDP6D. For the BMDP6D run, have the 6 continents identified on the bivariate plot by unique symbols (i.e., the letters A,B,C,D,E,F).

G. Using a chi-square test, test the independence of the FREEDOM variable with respect to the continents. Use both SPSS CROSSTABS and BMDP1F.

Computer Assignment No. 2
Due on class meeting number 7

For the Quality of Life data set, we want to predict the percent of the population in higher education (HIGHER) from selected variables. Using FREEDOM, GNP, and AGLAB as predictors, obtain solutions from SPSS REGRESSION and from BMDP2R. Obtain all residual listings and plots which are available from each program. By hand, use the BMDP output to compute (1) the squared part correlation (HIGHER,GNP.AGLAB. FREEDOM); and use the SPSS output to write the final, raw-score regression equation.

Computer Assignment No. 3
Due on class meeting number 8

A. Use the adult literacy variable, LITER, in the Quality of Life data set to predict BIRTHS. Obtain solutions, including all available residual listings and plots, with both SPSS REGRESSION and BMDP2R.

B. Create a variable equal to the square of LITER (in SPSS, use a COMPUTE statement; in BMDP, use a TRANSF statement) and redo the prediction of BIRTHS with a quadratic model. Set up a summary table showing sources due to linear and quadratic regression; compute the appropriate statistical tests, by hand, and interpret the results.

Computer Assignment No. 4
Due on class meeting number 11

A. The six continents in the Quality of Life data set represent a nominal variable. Create dummy variables for this factor; use SPSS REGRESSION and BMDP2R to undertake comparisons among the continents on adult literacy rates (LITER). By hand, use the SPSS output to obtain the mean on LITER for each continent using the regression coefficients and constant.

B. Create appropriate product variables and test for parallelness of slopes for the regression of LITER on BIRTHS using both SPSS REGRESSION and BMDP2R. By hand, use the BMDP output to obtain the regression equation for each continent using the regression coefficients and constant; and prepare an ordered regression summary table using the SPSS output.

NOTE: On SPSS runs, expand the workspace available for transformations; for example, start your run with

@SPSS*SPSS.SPSS,F 5000

Also, for the BMDP runs, in the REGRESSION paragraph, include the sentences ENTER=0.0. REMOVE=0.0 and TOLERANCE=.0001. to ensure that all variables enter the prediction equation.

Computer Assignment No. 5
Due on class meeting number 14

The nations of the 6 continents differ in adult literacy rates. Confirm this by running an analysis of variance using SPSS ONEWAY. In addition, do the following:

a. Enter VALUE LABELS and take OPTION 6 so that the continents are properly identified on the printout.

b. Using a .05 level of significance, run Tukey and Newman-Kuels pairwise comparisons. Explain why the results differ.

c. Prior considerations suggest the following contrasts:

NA versus SA
EUROPE and OCEANIA versus NA and SA
AFRICA versus average of all others

Test these contrasts with Bonferroni control of Type 1 error (i.e., use a significance level of .05/3 per test). Explain the basis upon which you reached each of these decisions.

Computer Assignment No. 6
Due on class meeting number 16

A. Afifi and Azen present data and analysis for a 2 x 4 design on pages 221-222. Confirm their results using both SPSS ANOVA and BMDP2V.

B. A three-way factorial design is presented by Afifi and Azen on pages 240-241. Confirm their analysis using both SPSS ANOVA and BMDP2V. The AB interaction is significant at conventional levels; present a relevant plot and interpret this interaction effect.

Computer Assignment No. 7
Due on class meeting number 18

The Quality of Life data set contains two categorical variables, Continent and Freedom Status. Treating these as factors in analysis of variance, use both SPSS ANOVA and BMDP2V to obtain results for the following criterion variables: HIGHER and LITER. Group the continents into a dichotomy: West (N. Amer. and S. Amer.) vs East (the others). For each dependent variable, produce a two-way table of cell means. Then, generate row and column means (a) by the simple average of the cell means, and (b) by weighted averages, weighting by the cell sizes. If the interaction effect is significant for either criterion variables, present an interaction plot of the cell means. using the SPSS output, produce the two ordered regression tables for each dependent variable which resulted in the sums of squares given. Note that the analyses will be "unbalanced". What effect does this have on the interpretation of the analyses of variance.

Computer Assignment No. 8
Due on class meeting number 20

Afifi and Azen present a four-group analysis of covariance design on page 267, with the analysis occurring on the next several pages of their text. Use both SPSS ANOVA and BMDP1V to reproduce their analysis. For the BMDP run, include contrast cards so that each of the 3 treatment groups is compared to the Control group. In addition, run the scattergrams for the groups with BMDP6D and sketch in the group-specific regression lines (by hand). Note that the BMDP program provides the homogeneity of regression test, but that SPSS lacks this feature. Produce the two ordered regression summary tables used by BMDP2V for the tests of means and slope (you will need to refer to the SPSS output in order to do this). Also, produce an ordered regression summary table conforming to the SPSS order, but including the test for equality of slopes; in this table, use a single denominator for all tests.

Computer Assignment No. 9
Due on class meeting number 22

The purpose of this analysis is to determine the contributions to predicting GNP of groups of variables from the Quality of Life data set. The predictor sets are:

A EXPECT, MORT, PHYS
B PRIM, HIGHER, LITER
C FREE/NOTFREE(d.v. with NOTFREE including partly)
D A by C Product Terms
E B by C Product Terms

Using SPSS REGRESSION, enter the sets of predictors in the above order (use EVEN inclusion levels to force each set in on one step). Then, by hand, set up an ANOVA summary table with appropriate statistical tests. Finally, re-do the assignment using this order: B, A, C, E, D. In all, you will have two regression orders and two summary tables.

Computer Assignment No. 10
Due on class meeting number 24

The problem is to compare, simultaneously, the Free and Non-Free nations on the basis of the 15 Quality of Life variables. The Non-Free group includes those nations which are Partly Free. Also, in this analysis, the continents are not used. Since the nations are currently categorized into 3 freedom groups, it is necessary, when using BMDP, to include a GROUP paragraph with a CUTPOINTS sentence: e.g., use 1.5 as a cutpoint to create two groups: those below 1.5 (i.e., the 1's) and those above 1.5 (i.e., the 2's and 3's). For the analysis, use BMDP3D and include the HOTELLING sentence in the TEST paragraph.

Computer Assignment No. 11
Due on class meeting number 25

The problem is to determine how well the 15 Quality of Life variables can discriminate among the nations in terms of the Freedom variable. Use BMDP7M to conduct a Discriminant Analysis with the grouping variable being the 3 Freedom groups and the criterion variables being the remaining 15 variables (excluding Continent). When setting up the DISCRIM paragraph, use the following sentences only: ENTER=0.0. REMOVE=0.0. JACKKNIFE.

Computer Assignment No. 12
Due on class meeting number 26

Once again using the Quality of Life data set, perform a Principal Component Analysis using the following variables: GNP, ENERGY, BIRTHS, DEATHS, EXPECT, MORT, PHYS, PRIM, HIGHER, and LITER. Conduct the analysis using BMDP4M. When setting up the FACTOR paragraph, use only the following sentences: METHOD=PCA. ROTATE=NONE. Also, include a PLOT paragraph with the sentence INITIAL=2. Repeat the analysis using SPSS FACTOR. With the FACTOR procedure, include the statements TYPE = PA1 and ROTATE = NOROTATE.

Note 1: Codes for continents are:
1 = N. Amer.
2 = S. Amer.
3 = Europe
4 = Asia
5 = Oceania
6 = Africa

Note 2: Freedom Status codes are:
1 = Free
2 = Partly Free
3 = Not Free

Variables are right-justified in their fields without decimal punched with the exception of population (cols. 6-10) and Annual Inflation Rate (cols. 26-30) which have the decimal punched. Thus, all variables may be read using an F5.0 format, except CONT and FREEDOM which must be read as F1.0. Missing data are represented by -99., except for Freedom Status (column 5) which has one missing case which is coded 9. The data appear on pages 134-136 of the Almanac cited above.

Table 1. Adjusted Means, Standard Deviations, and ANCOVA Results

Data Source	Computer Section N = 29		Control Section N = 17			
	Mean	S.D.	Mean	S.D.	F	Prob.
48 Common Examination Items	31.16	6.80	29.43	8.72	.77	.38
I have come to understand many useful statistical techniques in this course*	4.47	.51	3.73	1.10	11.38	.00
I learned many new skills which will benefit me in the future*	4.31	.55	3.76	1.16	6.85	.01
I had difficulty understanding the material presented in class**	2.90	1.25	2.05	.94	6.02	.02
This course was particularly well organized*	4.48	.74	3.59	1.23	9.64	.00
This course increased my interest in statistics*	3.98	.96	2.80	1.32	15.00	.00
I enjoyed my experience in this course*	3.68	1.00	2.55	1.42	11.81	.00
I had difficulty understanding the material presented in the textbook**	1.45	.73	2.30	.93	12.25	.00
I spent too much time and effort on this course**	2.45	1.30	2.23	1.19	.36	.60
I had adequate background preparation to take this course*	3.42	1.24	2.81	1.19	2.89	.10
The material in this course was covered too superficially**	3.81	.83	3.79	.85	.00	.95

*Scale is 1-5; 5 = agreement.
**Scale is 1-5; 1 = agreement.

Appendix

Description of the Data Set and Computer Assignments

The data set EDMS*STAT.QUALITY contains a number of variables which are related to the general quality of life within nations. The source of the data is the 1979 "Information Please Almanac." The names of the variables and the units of measurement are listed below.

Column	Code	Variable Name	Unit	Unit
1-3		Code # for Nation		
4	CONT	Continent	See Note 1, below	
5	FREEDOM	Freedom status	See Note 2, below	
6-10	POP	Population, 1976	Millions	
11-15	AREA	Area	Thousands of sq. km	
16-20	GNP	GNP per Capita, 1976	U.S. Dollars	
21-25	ENERGY	Energy Consump. per Capita, 1975	kg of coal equiv.	
26-30	INFL	Annual Inflation Rate, 1970-76	Percent	
31-35	AGLAB	Labor in Agriculture, 1970	Percent	
36-40	BIRTHS	Crude Birth Rate, 1975	per 1000 Population	
41-45	DEATHS	Crude Death Rate, 1975	per 1000 Population	
46-50	POP2	Proj. Population in 2000	Millions	
51-55	EXPECT	Life Expectancy at Birth, 1975	Years	
56-60	MORT	Infant Mortality, 1975	per 1000	
61-65	PHYS	Population per Physician, 1974		
66-70	PRIM	Persons in Elem. School, 1975	Percent of Age Group	
71-75	HIGHER	Persons in Higher Ed., 1975	Percent of 20-24 Population	
76-80	LITER	Adult Literacy Rate, 1974	Percent	

A COMPUTER-BASED TUTORIAL ON MATHEMATICAL INDUCTION*

by J. MACK ADAMS AND MARVIN LANDIS

Department of Computer Science
New Mexico State University
Las Cruces, New Mexico 88003

Abstract

The development of a tutorial on mathematical induction is described. The tutorial is based on the notion of informal verification of correctness of programs with one simple loop.

I. Introduction

This paper contains a description of a computer-based tutorial on mathematical induction and a discussion of the motivation for such a tutorial. It also describes the development of the tutorial, especially the problems encountered in the development.

The tutorial was authored in the latter stages of a project to develop computer-based learning material in computer science and mathematics. It represents an attempt to apply the techniques and expertise gained in preparing tutorials on lower level material, beginning programming and trigonometry [Mac81], to a topic that seems more difficult to teach.

II. Motivation and Approach

Motivation for developing the tutorial comes from a general difficulty encountered by the authors and their colleagues in teaching induction and, more specifically, from the following two problems:

The General Problem

The abstract nature of the usual approach to teaching induction does not seem effective for application-oriented students. More concrete examples than the usual formulas proved by induction seem necessary.

The Specific Problem

Using induction to teach proofs of program correctness [Alg78, Wir73] to students in upper division computer science courses is quite difficult, since these students are generally very application-oriented and seem not to have acquired a working knowledge of induction in their mathematical prerequisites.

The specific problem stated above led to an investigation of the following approach: reteaching induction using the students informal notions of program correctness as a relatively concrete basis and only then using induction as a tool in more formal proofs of correctness. Some success with this approach led to the following hypothesis:

Hypothesis

Concrete examples of informal justification of program correctness of programs involving one simple loop can be used as an effective basis for teaching mathematical induction to students with a knowledge of computer programming.

To test this hypothesis we decided to develop a computer-based tutorial that embodied the programming approach.

III. Authoring the Tutorial

The authoring followed the style developed at the Educational Technology Center at the University of California, which had proved effective in the development of previous tutorials for our project. Although we encountered difficulties that were much more challenging than those of previous tutorials, they did not have to do with deficiencies of the authoring style and thus no changes were necessary.

Before actually beginning the authoring we studied various approaches to teaching induction, particularly geometric approaches [Spe69, Wis70] since our material is designed for microcomputers with graphics capabilities. We also studied the origin of mathematical induction [Bus17, Caj18]. These studies provided good general background but no specific techniques or examples amenable to our approach.

We finally decided to use a relatively simple programming example in flowchart form. The example, given in Figure 1, was first used to introduce the notation and the idea of assertions at key points in the program. Then it was used for the informal verification of the output assertion.

*This paper is based upon work supported in part by the National Science Foundation under Grant No. SER-8005317.

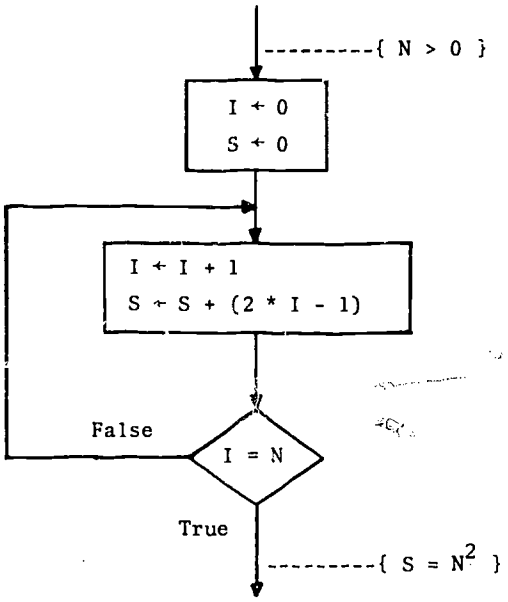


Figure 1

Simple Program with Input and Output Assertions

It thus provides a relatively concrete example of an induction argument.

The first major change in our preconceived notions about the approach came when we thought seriously about using the usual technique of a loop invariant, or inductive assertion, in the informal verification of correctness. We realized the degree of abstractness was only slightly less than that of traditional approaches to induction, and we recalled that students did not usually react well to loop invariants. In short, loop invariants do not seem strongly related to the, perhaps unconscious, process by which students justify the correctness of programs.

We eventually reached the conclusion that most students justify the correctness of a simple loop by:

1. tracing the simple cases of 1, 2, or 3 iterations, and then
2. tracing a general case by checking the final iteration, assuming that all has gone well in previous iterations.

On this basis we decided to:

a. motivate the basis step of induction by checking simple cases of a few iterations, and

b. motivate the induction step by checking the final iteration for an arbitrary value of an input parameter, N in Figure 1, with the explicit assumption that the algorithm has worked correctly to that point.

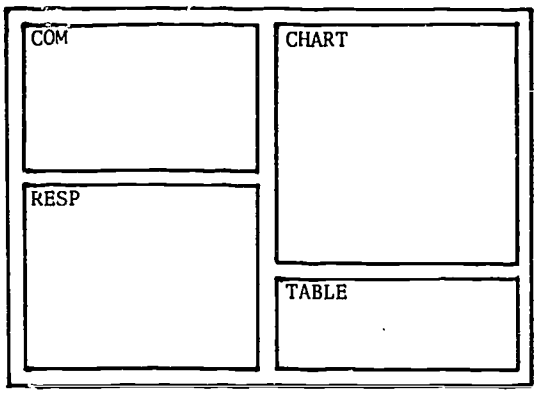


Figure 2

Port Layout

Essentially, we decided on an induction on the input parameter, rather than the usual induction on an "iteration variable".

After authoring the first version of the tutorial using the approach described above, the tutorial was presented to students and colleagues in a colloquium. One of the authors simulated execution of the tutorial using the blackboard as the screen, while the other author recorded audience response to questions posed within the tutorial and also suggestions for changes. This had not been done for our previous tutorials and we highly recommend it, particularly for tutorials on difficult subject matter. A second version was prepared, based on response from the colloquium, and a brief description of this version is given in the next section.

IV. Brief Description of the Tutorial

After displaying the title and credits, the simple program given in Figure 1, without the assertions, is displayed in the port named CHART (see Figure 2). The terminology of assertions is then introduced as the assertions are added to the display.

The student is then invited to verify, by computation, the output assertion for N = 1, 2, and 3. These results are recorded in a table in port TABLE.

At this point the student is asked to verify the output assertion for a value at the upper end of the table, and subsequently asked if he would like to verify the result when N is 209, a large value chosen arbitrarily by the authors. If the answer is "no", we heartily endorse the student's reluctance, and otherwise we question the advisability of taking on this task. This is used to motivate the need for some sort of general correctness argument.

The student then participates in the development of a correctness argument for an N of k+1,

assuming the algorithm worked when N was k and tracing another iteration. This is done in the following stages:

1. the output assertion when N is k is "backed-up" to an intermediate assertion, preceding the test, which we hope the student can correctly identify as being the same as the output assertion.

2. The intermediate assertion is then taken through the false branch of the test and the body of the loop, resulting in a new intermediate assertion and hence an output assertion for an N of k+1.

3. The student participates in the algebraic manipulation necessary to obtain the form of the output assertion upon which correctness has been based.

The resulting "induction step" is displayed for future use in port COM. This marks the beginning of severe screen space problems since we have only port RESP available for use.

At this point the student is asked to apply the induction step to the specific case when N is 3, which has previously been verified by computation. Thus he obtains correctness when N is 4 without computation of S. He is then asked to apply the induction step twice, starting again when N is 3, to obtain correctness when N is 5.

We then observe that the induction step is easier to apply starting from the value of 1 for N. This is a somewhat ungainly transition, but we felt it was desirable to first apply the induction step to obtain results for 4 and 5, which had not been previously verified by computation.

The student is then asked how many applications of the induction step are needed to verify correctness for a value of 209 for N, and subsequently for an arbitrary value, m, of N.

The process is then summarized in the form of a basis step, induction step and application of the induction step, and given the name "mathematical induction".

The length of the specifications for the version of the tutorial described above is approximately the same as those of our previously developed tutorials, so we expect the running time of the implementation to be about the same, 20-30 minutes. Implementation is now underway on the IBM PC using UCSD Pascal Version IV and our version of the packages Textport and Graphport obtained from the University of California at Irvine. We anticipate completion in time for trial usage of the tutorial in the latter half of the semester beginning in January of 1983. There seem to be no particular implementation problems except the one of screen space mentioned previously.

V. Summary

We have described the development of a tutorial on mathematical induction based on informal verification of correctness of a program with one simple loop. The resulting tutorial has also been summarized. We hope that relating the difficulties associated with the development of a tutorial over challenging subject matter may be helpful to others contemplating such developments.

We would like to recognize the valuable suggestions of colleagues and students, particularly those of Professor Warren Krueger and Mr. Jack Medd.

References

[Alg78] Alagic, Saud and Arbib, Michael A. The Design of Well-Structured and Correct Programs, Springer-Verlag, New York (1978).

[Bus17] Bussey, W. H. The origin of mathematical induction. The American Mathematical Monthly, XXIV, 5 (May 1917).

[Caj18] Cajori, Florian. Origin of the name "mathematical induction". The American Mathematical Monthly, XXV, 5 (May 1918).

[Mac81] MacKichan, Barry, Adams, J. Mack, and Hunter, Roger. Starting a computer based learning project. Proceedings of NECC 1981, National Education Computing Conference, Denton, Texas (June 1981).

[Spe69] Speck, Royce A. The number of squares on a goeboard. School Science and Mathematics (February 1969).

[Wir73] Wirth, Niklaus. Systematic Programming: An Introduction. Prentice-Hall, Englewood Cliffs, N.J. (1973).

[Wis70] Wiscamb, Margaret. A geometric introduction to mathematical induction. The Mathematics Teacher (May 1970).

IMPLICIT FUNCTIONS AND COMPUTER GRAPHICS

Sheldon P. Gordon

Suffolk Community College
Selden, NY 11784

Abstract

The present paper deals with the use of the computer to generate the graphs of implicit functions in the form $F(x,y) = 0$. In particular, several algorithms are discussed which can be used as the basis for a computer graphics program which will produce the graph of most such implicit functions. The value of having such programs available for use in both introductory and intermediate calculus classes is also discussed. Finally, a variety of examples are displayed to illustrate the results of such a program.

One of the least satisfying topics for students in calculus has to be that of implicit functions. Most students feel that there is nothing tangible for them to grab hold of with the topic. The old standby of performing algebraic manipulations is usually useless on all but the simplest expressions; for example, consider

$$x^4y + xy^3 = 1.$$

There is nothing to visualize in the way of a graph and no elementary method to solve for one of the variables in terms of the other. The fact that an Implicit Function Theorem guarantees the existence of such a function or its derivatives subject to appropriate sets of conditions is small consolation.

Fortunately, the power of sophisticated computer graphics now presents us with a tool which can draw the graph of most implicit functions. This capability allows us to add a valauble new dimension to the subject at both the elementary and intermediate calculus levels. At the introductory level, the availability of such a program provides the student with a better grasp of the concept of implicit function because he or she can actually "see" it. At the more advanced level, the underlying algorithm behind such a program can be as valuable as the program itself in teaching the students to analyze and anticipate the possible shapes of curves which can arise when dealing with implicit functions.

In the present paper, we will focus on an algorithm which will lead to the graph of the majority of implicit functions. We will also discuss some of the limitations and problems with such a program. Finally, we will demonstrate the use of one such program in producing the actual graphs of several implicit functions.

Suppose we start with an implicit function in the form

$$F(x,y) = 0.$$

Since this represents a functional relation with y as a function of x, then there should correspond (at least) one value of y for each value of x. In order to start the graph of such a function, it is necessary to determine one point on the curve. To accomplish this, we arbitrarily select a value for x, say x_0, and seek to locate a corresponding value y_0, by applying the Bisection Method to the equation $F(x_0,y) = 0$. We first locate the desired root by considering the sign for $F(x_0,y)$ on a sequence of intervals $[y, y+k]$ with sufficiently small k with y between -1000 and 1000, say. Once such an interval is found, the Bisection Method zeros in on the desired root fairly fast.

Now that a particular point (x_0,y_0) on the graph has been found (presumably), we seek to continue the graph by determining additional points and connecting them. Since the graph can continue both to the left and the right, we will have to consider both directions. Suppose we continue it to the right initially. We consider the sequence of points $x_i = x_0 + ih$ $(i=1,2,\ldots)$ and attempt to determine the corresponding values y_i. We could continue to apply the Bisection Method to generate the new points, but it converges far too slowly to use repeatedly. In addition, it also requires finding two values of y bracketing each successive solution y_i. Instead, we will apply Newton's Method in the form

$$y_{n+1} = y_n - F(x_i,y_n)/F_y(x_i,y_n)$$

where F_y is the partial derivative of F with respect to y.

With this approach, we note that each previously determined value y_i at (x_i,y_i) is used as the initial approximation to the next root y_{i+1} at (x_i+h, y_{i+1}). Since h is taken very small and the curve is presumably continuous and fairly smooth, y_{i+1} should be close to y_i and the convergence will usually be extremely rapid, if a solution indeed exists.

However, the "if" can be a big one. A great many oddities can occur when one deals with implicit functions and provisions must be made to account for them, if possible. First, Newton's Method breaks down when the denominator, $F_y(x_i,y_n)$, is (nearly) equal to zero. In the present instance, this corresponds to a vertical tangent which can take any of the forms illustrated in Figure 1. The cases shown in Figure 1a and 1b can be handled fairly easily using a simple modification of the above procedure. If the Method does not converge for a particular value of x, say $x_N = x_0 + Nh$, (as determined by either a count on the number of iterations being performed or on the relative changes in the values of the successive iterates), then it makes sense to skip on to the next point x_{N+1} while still using y_{N-1} as the initial estimate. If Newton's Method converges at this point within a reasonable number of iterations (certainly fewer than 10), then y_N can simply be set to the average of y_{N-1} and y_{N+1} for an acceptable graph.

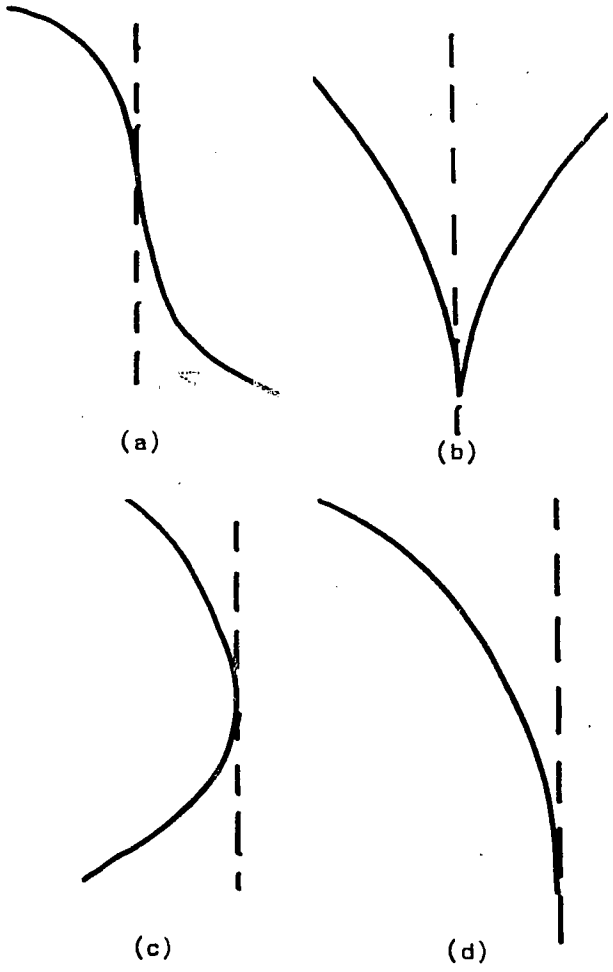


FIGURE 1.

As an alternative or even subsequent "fix" for the above problem, we can also resort to a secondary method. One such might be to call on the Bisection Method again. Another might be to use an extension of Newton's Method such as the one described by the present author in [1] which avoids the problem of $F_y = 0$ while developing a cubically convergent process based on approximating a curve with parabolas instead of tangent lines.

If all of these techniques fail, it is probable that the curve actually bends back on itself, as shown in Figure 1c. To attack this case, we have to locate a point on the second (or later) branch of the curve. This can probably best be done by recourse to the Bisection Method once again where we would have to take into account the likely orientation of the anticipated branch. Thus, if the y's are decreasing before the turn, then the search for the next point, also corresponding to $x = x_N$, should occur from $y = -1000$, say, up to $y = y_N$. Once such a point has been located, we simply set the step $h = -h$ and continue on to the left.

If this latter attempt also fails, we should probably just accept the inevitable and not try to extend the curve any further at that end. Instead, we should return to the original starting point (x_0,y_0) and attempt to extend the graph to the left using the same algorithm as above.

We note that the above efforts should essentially cover the case of a vertical asymptote as well. If the graph has the appearance shown in Figure 2a, then the continuation might be picked up with a "jump" across the singularity. If it looks like Figure 2b, then it is unlikely that the algorithm will converge quickly enough to have points on either side of the singularity connected. Alternatively, we can provide for such a singularity by keeping track of the relative sizes of the values for y and if they exceed some preset limit, then the graphing routine should allow for the "pen" to be lifted and reset across the jump.

There are several other possibilities which can also occur when one deals with implicit functions. Primarily, these involve the case of a

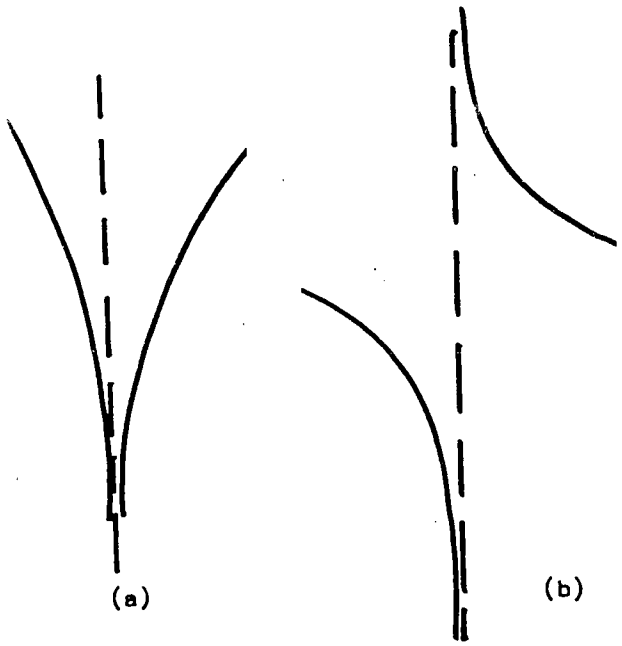


FIGURE 2.

horizontal cusp, as shown in Figure 3. Both of these cases should be handled by the procedure discussed above when the curve turns back onto a new branch. However, if the curve bifurcates at a particular point, then the algorithm will continue along just one of the branches and it does not seem possible to design any reasonable modification of the algorithm which can pick up such a situation.

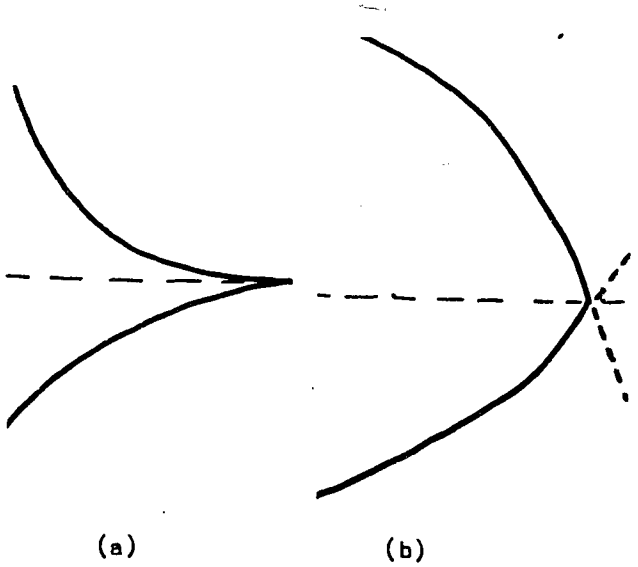


FIGURE 3.

Another major problem, which appears to be insurmountable in terms of any reasonable modification of the present method, involves the situation where the curve consists of two or more distinct and non-intersecting segments. For example, consider the case of the simple hyperbola. The algorithm will trace out one portion of such a curve, but will miss entirely the other portions. Further, any effort to take such an eventuality into account would seem to make the resulting program incredibly unwieldy. Consequently, we will ignore this case and hope that it does not arise or simply be content with tracing out a single portion of the curve.

A flow chart for the above algorithm is presented in Figure 4. From this, it should not be too difficult to write the corresponding computer graphics program to implement the method on most of the available high resolution computers. The author has already done so for a PDP 11/34 system supporting Tektronix graphics terminals (model 4006) as well as for the TRS 80 Color Computer. However, it should be emphasized that the actual procedure is an extremely time-consuming one, particular if a reasonably large number of points are to be located. In fact, the amount of number crunching is probably comparable to that needed to graph three dimensional surfaces. As such, it is not well suited for microcomputers on a real-time basis unless one is using a preprocessor to handle the calculations. Alternatively, it is possible to generate such graphs in advance and "record" the images for later demonstrations. Unfortunately, this certainly involves a tremendous loss in spontaneity and a consequent reduction in impact on a class.

It is worth noting that there is an alternate algorithm which also works quite well in producing the graphs of most implicit functions. In fact, the author has found that it is usually more effective than the one presented above. In particular, the idea is to transform the given function $F(x,y) = 0$ into polar coordinates in the form

$$F(r\cos\theta, r\sin\theta) = 0$$

and graph it with r as the implicit function of θ. A natural choice for a starting point might be $\theta = 0$ and a full loop will often occur for θ between 0 and 2π. One further advantage is that most of the shapes shown in Figures 1, 2 and 3 could be traversed smoothly in polar coordinates while they present major difficulties in rectangular coordinates. One potential problem does arise with a vertical cusp and a procedure for handling this would be analogous to the method used earlier when a rectangular curve bends back on itself. Another advantage of the polar representation is that it would be often easier to check for a complete loop of a closed curve (simply compare r for $\theta = 0$ and for $\theta = 2\pi$, say.)

There are several further complications that occasionally arise, particularly with the polar representation. For one, if we utilize a function involving the trigonometric functions especially, then there are inherent difficulties built into

the use of Newton's Method in terms of instability. If we have a point on such a curve anywhere near (and that can be quite far, relatively) a root of the derivative, then the corresponding tangent line will shoot off to a point far distant. As a result, Newton's Method can easily converge to a succession of distant point[illegible] di[illegible]nt branches of the curve and the resulting graph, when these points are connected, will be highly unreliable, albeit quite interesting in shape.

A second difficulty is a computational one. In order to pick up the initial point, a search method has been suggested to bracket the value, involving a large range of values for y or for r. However, if the given function involves exponential terms such as EXP(Y), then the capacity of the computer can easily be exceeded causing an overflow error. Similarly, an all-purpose program applied to a general expression F(x,y) = 0 cannot anticipate all possible instances where a function is not defined - say if LOG(Y) occurs or if fractional exponents are involved.

In view of these comments, it should be clear that while this type of program can be an exciting and instructive one for students, it has to be used with considerable advanced planning or a willingness to encounter errors with good humor.

Incidentally, it might appear strange that the major emphasis above was devoted to the rectangular case despite the fact that the polar form is usually more effective and easier to apply. The author feels, though, that from a purely pedagogical point of view, the rectangular approach appears far more natural to the students. Almost all problems in calculus dealing with implicit functions are in rectangular form and, in fact, in introductory calculus, the student encounters implicit functions long before he or she sees polar coordinates. Thus, if the student is to see the program rather than simply the output, then the use of the rectangular form is to be preferred. This is probably even more important in intermediate calculus where the emphasis is likely to be placed more on the possible behavior of such functions than on the actual appearance of some of them.

Finally, we illustrate the use of such a program to produce the graphs of a variety of implicit functions. In Figure 5, the graph of $(x+y)^3 + (x-y)^3 = x^4 + y^4$ is shown. In Figure 6, the implicit function displayed is given by $x^5 + 4xy^3 - 3y^5 = 2$, while in Figure 7, we show the graph of $x^7 - 4x^3y^2 + 4y^6 = 1$. Lastly, in Figure 8, we show the graph of the implicit function $xy^4 - \sin(xy) + x = 1$. Once such graphs are available, an immediate followup would be to investigate the results of modifying the values for the constants on the right to examine the corresponding families of curves so generated. However, we will not do any of this here.

Reference:
Gordon, S. P. and E. Von Escher, On a cubic extension of Newton's Method. (submitted).

Acknowledgement:
The author gratefully acknowledges the support provided in part for the developments discussed in the present paper by the State University of New York under a grant from its program for Improvements in Undergraduate Instruction.

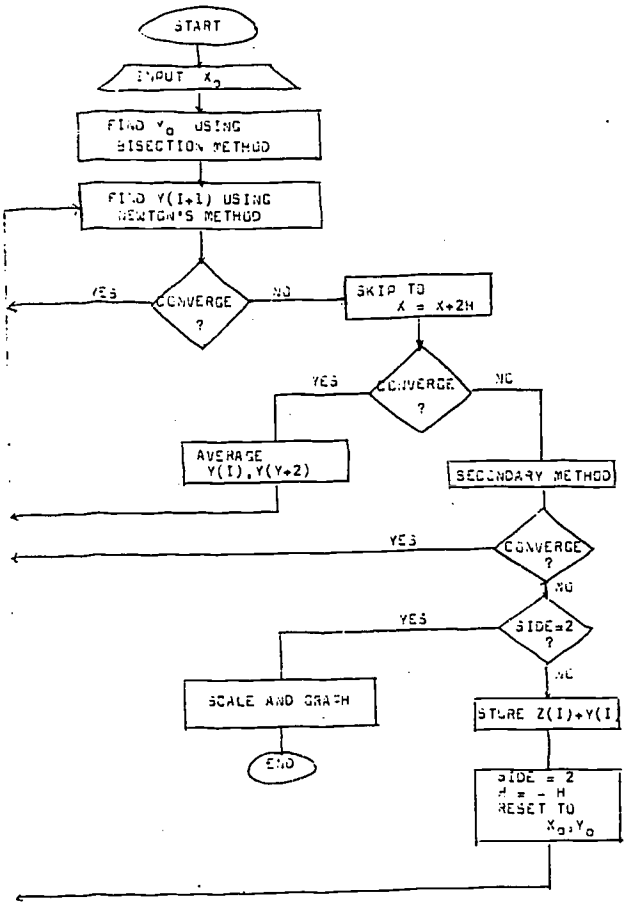
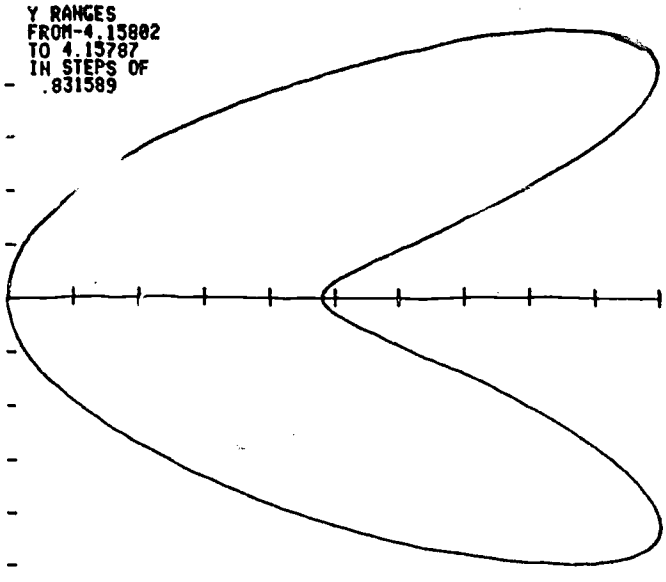


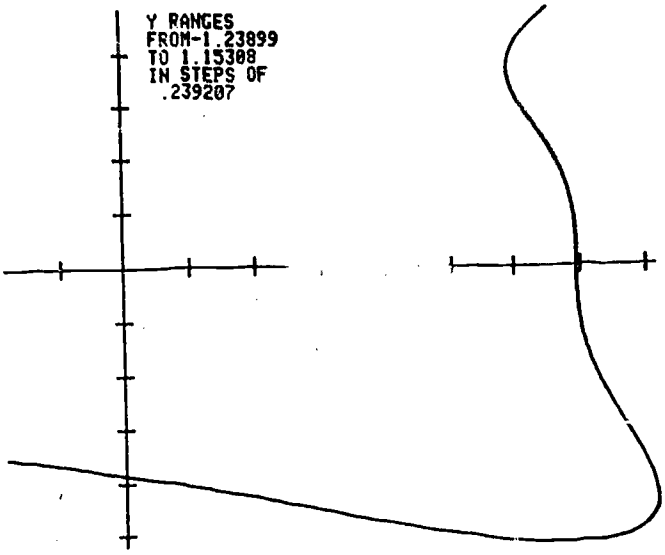
Figure 4.

365



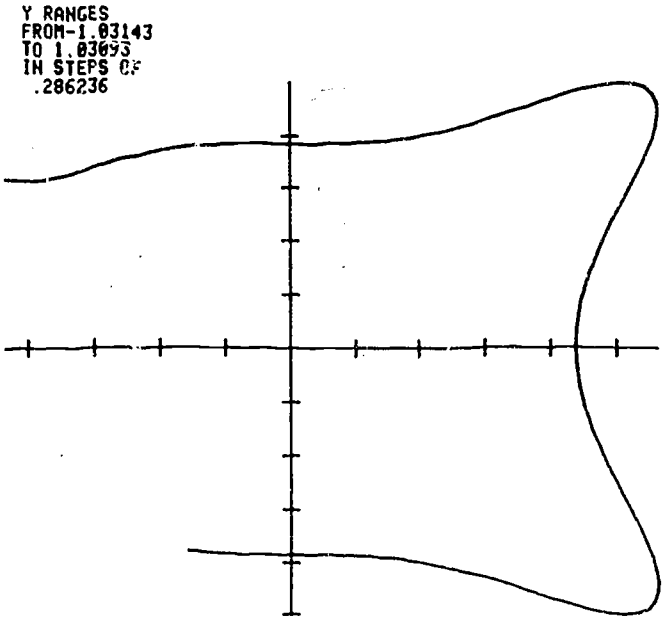
X RANGES FROM .924778E-6 TO 4.16212 IN STEPS OF .416212

FIGURE 5: $(X+Y)^3 + (X-Y)^3 = X^4 + Y^4$



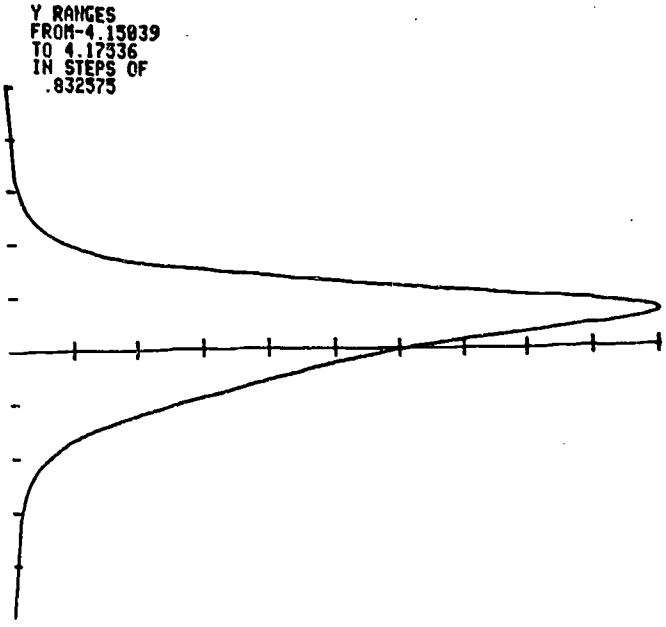
X RANGES FROM-.306732 TO 1.34491 IN STEPS OF .165165

FIGURE 6: $X^5 + 4XY^3 - 3Y^5 = 2$



X RANGES FROM-.993599 TO 1.28314 IN STEPS OF .227674

FIGURE 7: $X^7 - 4X^3Y^2 + 4Y^6 = 1$



X RANGES FROM .330507E-2 TO 1.63301 IN STEPS OF .16297

FIGURE 8: $XY^4 - SIN(XY) + X = 1$

INTERRUPT DRIVEN I/O PROJECTS IN AN ACM '78 CS4 COURSE*

by Greg Starling

Department of Mathematics and Computer Science
Western Carolina University
Cullowhee, North Carolina 28723

Abstract

In the ACM '78 curriculum recommendations for CS4, Introduction to Computer Organization, the three objectives of the course involve some aspect of simple input/output devices at both the software and hardware level. These recommendations also call for the study of a simple minicomputer or microcomputer system.

In a computer science program which is oriented toward a time-sharing facility some objectives of the course are impossible to meet. Even with microcomputers some of the aspects of I/O programming do not come alive without a facility for binary input and output ports with interrupt capabilities. We describe some hardware and software I/O projects which we use in our version of CS4. Included in the paper are details to allow others to implement these projects. We use APPLE II plus microcomputers, but the principles could be adapted for other microcomputers.

Introduction

An abbreviated description of the ACM Curriculum '78 CS4, Introduction to Computer Organization, course is:

Course Objectives.

(a) to introduce the organization and structuring of the major hardware components of computers;
(b) to understand the mechanics of information transfer and control within a digital computer system; and
(c) to provide the fundamentals of logic design.

Topics.

(a) Basic Logic Design
(b) Coding (BCD, ASCII, etc.)
(c) Number Representation and Arithmetic
(d) Computer Architecture
(e) Example (an actual microcomputer system)

At Western Carolina University we are now in our fourth year of teaching this course. In the 1979-80 academic year we used a Z-80 based microcomputer (Exidy Sorcerer and TRS-80) to support the hardware and programming topics of the course. In 1980-81 we established a microcomputer laboratory with 15 6502 based APPLE II plus microcomputers and a CORVUS Systems hard disk network storage facility under NSF Grant number SER-80-04761. In both years we found that some of the topics dealing with input, output and logic design were rather artificial without some means of non-keyboard, non-printer or CRT I/O. In fact, the concept of interrupt driven systems remained a mystery to most students, even some of the best ones. In 1981-82 it was decided during the logic design and memory organization phases of the course to design a binary input/output port with interrupt capability using 6520 PIAs (Peripheral Interface Adaptors). After studying logic design, memory mapped I/O and the organization of the 6502 and 6520 chips the class, under the guidance of the instructor, established the design criteria and designed the memory address decoder to place the I/O port in a peripheral expansion slot of an APPLE II computer. The instructor then did the mechanical design and fabrication of the I/O device. The students then wrote program exercises using the port.

Hardware.

The design criteria decided upon were:

(a) Two bytes for input and two bytes for output to facilitate easy transfer of signed numbers as large as 32,767 in magnitude and to provide both a status register and a data register for programmed I/O experiments. This requires two PIAs.

(b) Toggle switches for input and LEDs (light-emitting diodes) for output so that the relationship between characters on the video screen or keyboard can be easily observed by way of the I/O port.

(c) Momentary pushbutton switches and LEDs for interrupt requests and interrupt acknowledge signals using the peripheral input lines and peripheral control lines of the PIAs. It was agreed that each byte of the input port and each byte of the output port would have an interrupt request button and acknowledge LED. One button would have normally open contacts and one would have normally closed contacts to provide the possibility of an

* This material is based upon work supported by the National Science Foundation under Grant No. SPE-82-63111. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not reflect the views of the National Science Foundation.

interrupt on falling edge or on a rising edge signal.

(d) A slope-front, desktop console to mount the switches and LEDs for ease of use while controlling the computer from its keyboard.

(e) An edge-card contacts circuit board which fits an APPLE computer peripheral expansion slot to hold the PIAs and address decoding logic. (Soldering should not be done by a novice with a soldering iron. Wire-wrapping is probably the best choice, but use one level wire-wrap sockets so as not to interfere with other peripheral cards inserted in adjacent slots.)

(f) Use a 50 conductor ribbon cable (2' - 3' long) and IDC (Insulation Displacement Connectors) to interconnect the console and the card.

Each of the eight peripheral slots on the APPLE Computer's expansion bus is allotted sixteen memory mapped I/O locations beginning at location $ C Ø 8 Ø ending at location $ C Ø F F. For slot k the address range is $ C Ø n Ø - $ C Ø n F, where n = $ k + 8. In addition, each of slots one through seven has a 256 byte page of memory allocated beginning at $ C 1 Ø Ø and ending at $ C 7 F Ø (refer to chapter 5 of the APPLE II Reference Manual[1]). There is also a 2K byte block of ROM space reserved for the use of all of the peripheral slots so that service routines longer than 256 bytes can be accommodated. Although we did not implement either a 256 bytes primary page or 2K byte expansion block in the class, we did study their use and one student implemented both in a one hour summer school special topics course. The EPROMs are visible in figure 4.

The details of the hardware design are contained in the following tables, figures and diagrams. Table 1 and Figure 2 give the pertinent architectural and organizational details of the 6520 PIA. Figure 3 contains information on interfacing the PIA s to the APPLE bus. For more details than these, the Motorola data sheets for the 6520 (same as 6820) PIA and the APPLE II Reference Manual [1] should be consulted.

Address Lines		Control Register Bit X = Don't Care		Register Selected
RS1	RS0	CRA-2	CRB-2	
0	0	1	X	Peripheral Register A
0	0	0	X	Data Direction Register A
0	1	X	X	Control Register A
1	0	X	1	Peripheral Register B
1	0	X	0	Data Direction Register B
1	1	X	X	Control Register B

Table 1. PIA configuration information

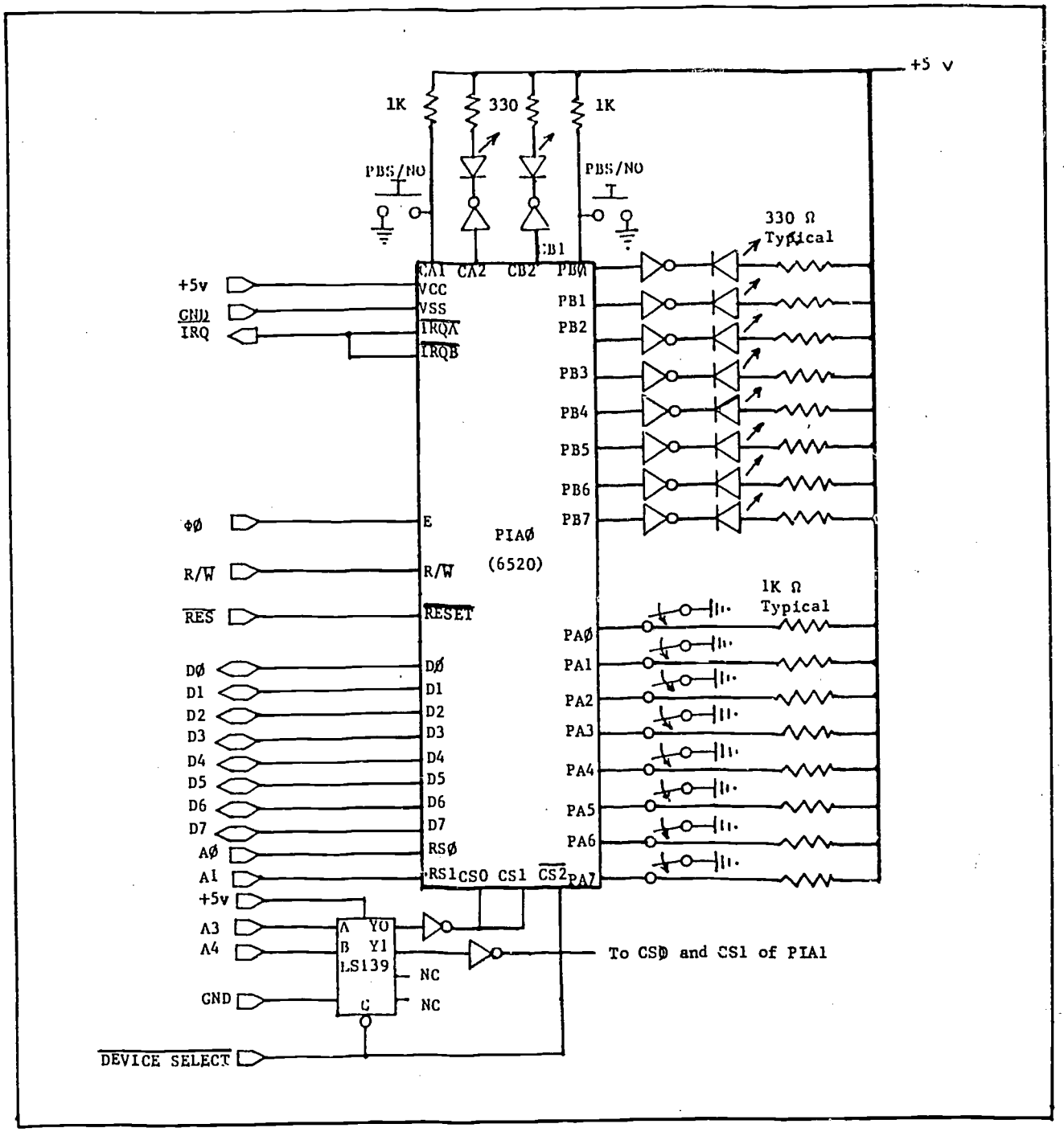
	7	6	5	4	3	2	1	0
CRA	IRQA1	IRQA2	CA2 Control			DDRA Access	CA1 Control	
CRB	IRQB1	IRQB2	CB2 Control			DDRB Access	CB1 Control	

Notes.

Data Direction Access Control Bit (CRA-2, CRB-2). Bit 2 in each control register (CRA, CRB) allows selection of either a Peripheral Interface Register or the Data Direction Register when the proper register select signals are applied to RS0 and RS1.

Interrupt Flags (CRA-7, CRA-6, CRB-7, CRB-6). These flags are set by active transitions on CA1, CA2, CB1, CB2 respectively. They are reset by peripheral read data operations.

Figure 2. PIA Control Registers (CA1,CA2,CB1,CB2 control)



Notes.

1. All resistors are in 10 pin, 9 resistor SIP packs.
2. Inverters are 74LS04.
3. LEDs are T1 3/4 red
4. Push button switches are NC (normally closed) on PIA1.
5. Toggle switches are SPDT, long handled, miniature.
6. LEDs and inverters on outputs are powered from an external +5 v supply.

Figure 3. Typical connections for one of the PIAs

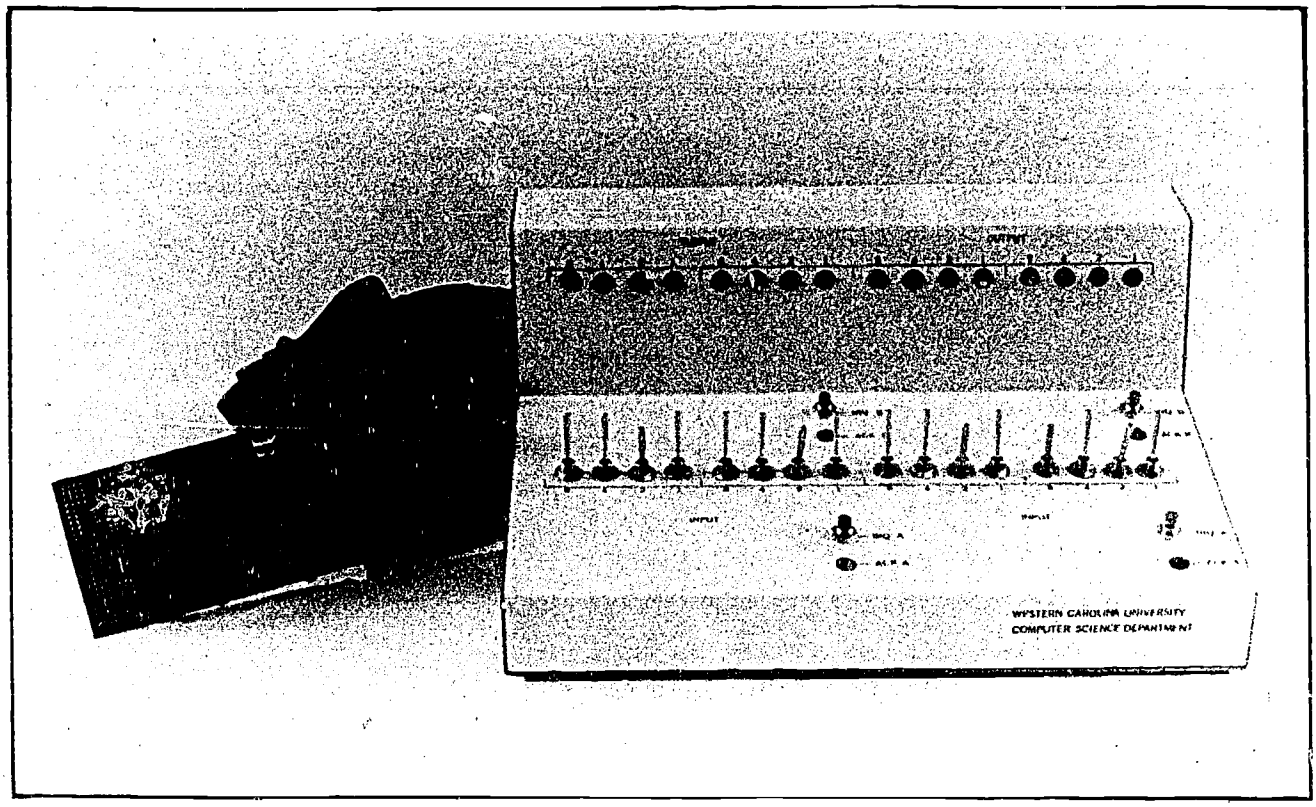


Figure 4. The finished product

Software Projects.

Many of the programming exercises in an assembly language programming course at the college sophomore level are repetitions of the exercises done in the beginning programming course in the freshman year. Usually, only those exercises involving code conversions are significantly different from those done in the first course. That is because traditional assembly language programming courses have been taught on mainframe, central, time-shared computers. This has prohibited the undertaking of most basic I/O and interrupt programming. In fact, in most systems the students must be taught how to interface with the system's I/O routines just to get input to and output from their assembly language programs. There is no such thing as raw binary input and output. And as far as interrupt programming is concerned there is none. These are two deprivations that insulate computer students from some of the most exciting capabilities of computers. Fortunately, the growth of microcomputer usage in the introductory computer organization course gives us the possibility to rectify these shortcomings. With the I/O port described here attached to an APPLE microcomputer both I/O programming and interrupt programming exercises are easily undertaken. Indeed, basic data transmission programming, an increasingly more important part of programming, is more easily understood.

Some of the programming assignments we use with the binary I/O port are as follows.

(1) Configure the PIAs in the I/O port. (See chapters 11 and 12 of 6502 Assembly Language Programming[3].) This gives quite a dramatic effect, because the LEDs are all lit on power up, and all go out upon successful configuration.

(2) Load the APPLE's accumulator from the input switches and transfer the contents to the output port. This may be done for both an eight bit transfer or a sixteen bit transfer.

(3) Do exercise (2) in an unending loop so that the output changes continuously with the input (flip a switch on and the corresponding LED lights).

(4) Take a character from the keyboard and transfer it to the output lights. The COUT subroutine in the APPLE's Monitor or Autostart ROM is useful for this exercise. (See page 61 of the APPLE II Reference Manual[1].)

(5) Take a code from the input switches and transfer it to the APPLE's video screen either as an ASCII character or as hexadecimal digits. The subroutines PRBYTE, PRHEX, PRNTAX, and RDKEY (see pages 61-62 of the APPLE II Reference Manual[1]) are used in these exercises.

(6) Take a binary number from input switches, rotate or shift, display on the LEDs and repeat after an appropriate time delay so that the bits move slowly enough to observe. The APPLE's WAIT subroutine (see page 63 of the APPLE II Reference Manual[1]) is useful here.

(7) Take two 8-15 bit numbers from the switches, add them and display the result on the LEDs and the screen. The sixteenth switch is used

as a strobe bit to signal the program that an operand is set on the switches. This gives experience on programmed or polled input status checking as opposed to interrupt driven input acceptance.

(8) Configure the control registers of the PIA s for interrupts (see figure 3). The control registers contain interrupt enable bits (CRA-0, CRB-0, CRA-3, CRB-3) interrupt status bits (CRA-7, CRB-7, CRA-6, CRB-6) direction select bits (CRA-4, CRB-4) and active transition bits (CRA-4, CRB-4). For more details see pages 11-15 through 11-20 of 6502 Assembly Language Programming[3].

(9) Configure the two PIA s so that the binary I/O port is two 8 bit input/output ports with CA1 interrupts enabled on both PIA s. Modify exercise (3) so that input/output port 0 and input/output port 1 can interrupt each other. Input and corresponding output comes from first one and then the other depending upon which one had its IRQ button pressed last. An added feature to this exercise would be to light the ACK LED of the active port. This is controlled by making CA2 an output signal on both PIAs. We have not tried this particular feature, but we believe it should work. The program might be written to poll the status bits (CRA-7) on the two ports to determine which issued the IRQ.

There are several cautions about interrupts on the 6502 MPU and the 6520 PIA. Upon responding to an interrupt the 6502 disables interrupts, so to make it possible for the other port to interrupt an interrupt service routine bit 2 of the P (flags) register should be reset. Also the service routine should read the data register of the port sending the IRQ to reset the interrupt status bit, CRA-7 and thus deactivate the IRQ for that port.

When the interrupt service routine is loaded into the APPLE's RAM one must also load the address of its first executable statement into locations $3FE and $3FF. An IRQ causes an eventual jump to the location stored there. This is because the locations $FFFE and $FFFF which hold the true interrupt vector, $FA4Ø, are in ROM space on the APPLE II. On interrupt request execution is vectored to that location which eventually leads to an indirect jump to the location whose address was stored at $Ø3FE - $Ø3FF. Refer to page 143 of the APPLE II Reference Manual.[1] If one were to implement the page of memory allocated to an APPLE expansion slot, then the service routine could be located in there. If the routine is too large for this 256 byte page, then one would have to include a 2K expansion ROM (or RAM) so that the service routine could jump out of the single page into that 2K of memory. This 2K of address space is reserved so that any board plugged into an expansion slot can switch its own 2K of memory into it. The proper protocol for this substitution can be found on pages 84-85 of the APPLE II Reference Manual.[1] We provided a unique decoding of the special location address $CFFF. One student who took the CS4 course has subsequently done that. He put configuration routines for the PIAs and some of the exercise routines listed above into 2716 EPROMS. We actually used 2716 EPROMS both for the single page and for the 2K expansion ROM space because our EPROM programmer doesn't program the 256 byte 1702 EPROMS.

Listing 1 configures the PIAs for APPLE peripheral slot number 2 and transfers the input switches values to the output LEDS.

Listing 1.

```
$9500    LDA #0
$9502    STA $C0A1     ; SELECT
$9505    STA $C0A5     ; DATA
$9508    STA $C0A3     ; DIRECTION
$950B    STA $C0A7     ; REGISTERS
$950E    STA $C0A0     ; DATA REGISTERS
$9511    STA $C0A4     ; A WILL BE INPUT
$9514    LDA #$FF
$951A    STA $C0A6     ; B WILL BE OUTPUT
$951D    LDA #04
$951F    STA $C0A1     ; SELECT
$9522    STA $C0A5     ; DATA
$9525    STA $C0A3     ; REGISTERS
$9528    STA $C0A7
$952B    LDA $C0A0     ; READ LOWBYTE SWITCHES
$952D    STA $C0A2     ; TRANSFER TO LOWBYTE LEDS
$9531    LDA $C0A4
$9533    STA $C0A6     ; TRANSFER TO HIGH BYTE
                       ; LEDS
         STA $C0A6     ; TRANSFER TO HIGH BYTE
                       ; LEDS
         JMP $952B     ; REPEAT FOREVER (UNTIL
                       ; RESET)
```

Summary.

The binary I/O device provides the means for very basic input/output and interrupt programming exercises which are not traditionally available to computer science students. With it we believe that students get a better understanding of code conversions, the I/O interface and interrupt driven systems. It prepares them better for the required upper division courses and also makes it possible for them to start at a higher level in some of the hardware oriented elective courses.

Plans for the future are to continue the projects but to change from the PIAs to a VIA (Versatile Interface Adaptor) because it has all the features of the PIA, two interval timers and a serial output in addition. It will allow us to have timed interrupts exercises which will be useful in studying time-sharing in a systems programming course.

REFERENCES.

1. APPLE II Reference Manual, Copyrighted 1979, 1981 by APPLE COMPUTER INC.

2. "Add a Peripheral Interface Adapter to Your Apple II," by Kenneth J. Ciszewski, BYTE Magazine, January 1982.

3. 6502 Assembly Language Programming, by Lance A. Leventhal, Osborne/McGraw-Hill, 1979.

ASSEMBLY LANGUAGE ON THE APPLE -- A THOROUGH INTRODUCTION

W. D. Maurer, Professor
The George Washington University (SEAS)
Washington, D. C. 20052

Abstract

An increasing number of colleges and universities are setting up APPLE computer laboratories. These are being used for the teaching of BASIC, as well as for experimental projects in a wide variety of courses. Assembly language, however, does not seem at this time to be widely taught on the APPLE. One reason for this may be that until recently the APPLE Corporation did not produce a good assembler of their own. One was forced to go to other organizations, such as Lazer Systems, which act as distributors for assemblers written independently. The present paper outlines the results of a project involving the preparation of a thorough syllabus for the teaching of assembly language on the APPLE. As a measure of the success of this effort, the department in which this author teaches has just instituted a new course in assembly language programming of microcomputers, to be offered as an alternative to the study of IBM 370 assembly language.

Introduction

There are many arguments for teaching assembly language on a microcomputer such as the APPLE. Of course, there is the basic argument concerning progress in computer science; microcomputers provide better throughput (instructions per dollar) than mainframes, and therefore constitute an advance in computer science that should be reflected in curricula. But there are other arguments. Assembly language programming on computers such as the IBM 4341 has been strongly discouraged for some time now. Assembly language programming on microcomputers, on the other hand, is often necessary because of the small memory size of the microcomputer system being constructed.

It may be argued that microcomputer assembly language is of particular importance in a compiler-writing course, since most of the new compilers being written these days are being written for microcomputers. Many new assembly language concepts are available in microcomputer assembly languages which are not available on the IBM 4341; these include stack-oriented call and return instructions, push and pull (= pop) instructions, and status flags (an improvement on condition codes). Even some more traditional assembly language concepts, such as immediate addressing for comparing and subtraction, indirect addressing, and direct memory increment, decrement, and shift, are available on the APPLE and not on the 4341. Finally, there is the sheer challenge of working on a micro -- learning to do without multiply and divide, without floating point, and, in the case of the 6502, without 16-bit operations or add-without-carry.

Existing Literature

The existing literature on the assembly language of the 6502 (the particular microprocessor used in the APPLE) was quickly found wanting. There are several short books on APPLE assembly language, many of which have no exercises, and none of which was felt to be thorough enough. (Our syllabus, which runs to over 500 manuscript pages, has, however, been accepted for publication [1] by Computer Science Press, Rockville, Md.) There is also a very throuugh book on the assembly language of the 6502 [2]. This book, however, is intended for the teaching of logic design replacement. As a consequence of this, the student, in using it, does not require a knowledge of BASIC, but does require an elementary knowledge of computer hardware (equivalent to Volume I of [3], for example), whereas the reverse is true in a typical assembly language programming course as it is given in colleges and universities (and in our syllabus).

In teaching a course on APPLE assembly language, it is essential to specify an assembler. We use the Lazer Systems Interactive Symbolic Assembler (LISA), for several reasons. It is one of the few privately produced assemblers for the 6502 to be systematically maintained, down through the years; new versions continue to be produced, involving new features as well as correction of problems discovered in older features. It is very fast, being what used to be called an "in-core" assembler; that is, assembly is directly into memory rather than to a diskette

file. It has consistently been rated as the best APPLE assembler, overall, in the evaluations that we have seen. Finally, it is reasonably priced.

Our APPLE laboratory consists of 12 APPLE II+ systems and eight APPLE III systems. At the moment LISA runs only on the APPLE II+, and PASCAL runs only on the APPLE III (and on one of the APPLE II+ systems, which has the Language Card). Hence PASCAL classes are scheduled for the IIIs, LISA classes for the II+ systems, and BASIC classes for either machine. In the fall of 1982, six sections of assembly language were run, of which four were oriented towards the 6502.

Description of the Syllabus

A heavy emphasis is placed on written exercises. In the teaching of assembly language, one finds far more concepts that have to be learned than is the case with FORTRAN, BASIC, or even PASCAL. Some of these can be learned by hands-on experience in assembly language programming, but there is a limit to the amount of material that can be learned in one semester in this way. Accordingly, we have arranged the syllabus in 100 small sections (a maximum of three manuscript pages per section), with three written exercises per section, each of which has no more than three parts. These have been assigned to the students as homework in each of three semesters (spring, summer, and fall 1982), and checked very thoroughly, both as to possible incorrect answers supplied by the instructor (who makes more mistakes than he cares to admit) and as to difficulty (a number of the exercises were revised and made easier). Several students submitted acceptable alternative answers, which have been included in the section on answers to selected exercises.

The student is introduced to the computer at a much later time, during the semester, than is customary in programming courses. Assembly language differs from other programming languages in that a tremendous amount of material must be understood before the student is capable of writing reasonable programs. It is possible to write a program which adds two numbers, for example, in the first week, but this, in our opinion, does not facilitate learning how to write and debug more sophisticated programs. In the outline of the syllabus, given below, it is made clear how much material we cover (about two-fifths of the total) before the first actual hands-on programming assignment.

In quite a number of texts on assembly language, the student is presented, at the start of the course, with a complete list of all the instructions of the given machine, and this is followed by a number of chapters on specific programming techniques. In our syllabus, we have taken a different approach. The introduction of the various instructions, status flags, registers, and addressing modes is spread out over the whole syllabus. Thus the PSW (the status register), for example, is not introduced until section 67; the interrupt status flag and the last three instructions (CLI, SEI, and RTI) are introduced in section 68; and the last three addressing modes (zero-page and the two kinds of indirect indexed addressing) are saved for sections 74 through 76. Our motivation here is to provide a thorough introduction to every separate instruction, register, status flag, and addressing mode. Thus there is a separate section on the logical exclusive OR, for example, giving several applications, and introducing the truth table and the two alternate mnemonics (EOR and XOR). Similarly, each status flag is introduced in a section of its own, together with a discussion of its uses. (The uses of the carry flag are spread over at least a dozen sections.)

Detailed Outline of the Syllabus

(1) Codes (Section 1). The basic idea of representing all data in a computer as codes is introduced by analogy to the secret codes which children pass back and forth.

(2) Number systems (Sections 2-5). An introduction to binary and hexadecimal, conversions between one number system and another, and addition and subtraction in binary and hexadecimal.

(3) Registers (Sections 6-7). The basic ideas of registers and memory cells; the A, X, and Y registers (the others are introduced later); the basic fact that n bits can hold 2^n codes; multibyte quantities and twos' complement arithmetic.

(4) Load, store, increment, and decrement (Sections 8-9). The first twelve instructions: LDA, LDX, LDY, STA, STX, STY, INC, INX, INY, DEC, DEX, and DEY. (We do not introduce ADC or SBC until later, because of the need to discuss carrying and borrowing.)

(5) Machine and assembly language, and pseudo-operations (Sections 10-11). At this early stage, the student is shown, in a simple form, the difference between assembly language and machine language, and is introduced to three pseudo-operations: ORG, END, and DFS. These will be enough for the writing of simple programs. (We may note that many syllabi start out with machine language, and only later introduce assembly language, hoping to make the point that the tediousness of machine language suggests the need for assemblers. We take a different tack here because we see no need to make this subject any more tedious -- there are enough fine points to learn as it

is.)

(6) Two-byte quantities (Section 12). We are going to be doing 16-bit operations throughout the syllabus (as is clear from what follows). At this point our purpose is merely to introduce the notation K+1 (as in LDA K+1) for the upper byte of the two-byte quantity K, and to start clearing away the common misimpressions about K+1 (distinguishing LDX K+1 from LDX K followed by INX, for example).

(7) Indexing (Section 13). An unusual note here: this early in the syllabus, we in roduce LDX J followed by LDA T,X for the loading of T(J) into the A register. It is our strong opinion that subscripted variables are usually introduced far too late in the semester, regardless of the programming language being taught. One must face the fact that over 95% of all programs contain arrays and indexing.

(8) Adding and subtracting (Sections 14-17.) This includes 8-bit and 16-bit addition and subtraction, the carry flag, and the relation between carry and borrow. Two-byte numbers are considered as two-digit numbers in a number system with base 256, so that their addition and subtraction can be seen to follow the same rules as with other number systems. We may also note that a careful and precise treatment of the carry flag as used in subtraction, specifically, is markedly absent from many treatments of microcomputer machine language programming.

(9) Transfer instructions and comments (Section 18). Two topics here: first of all, four more instructions, TAX, TXA, TAY, and TYA, and their use in the evaluation of expressions such as T(J+K) (where T is an array); then, comments (which follow a semicolon, if the LISA assembler is used). The reason for introducing comments at this point is that our programs are just now starting to get big enough that the student can see the need for some memory aid to remember what has been done.

(10) Branching and labels (Section 19). Since we have introduced the carry flag, we can introduce BCC and BCS, and this leads naturally to a discussion of the syntax of labels. Here the student who has had BASIC or FORTRAN needs to get used to the idea of an alphanumeric label. An application of BCC and BCS (adding or subtracting two quantities, one eight bits long and the other 16) is also given.

(11) Loops and zero status (Sections 20-22). The student who has had only FOR statements in BASIC, or DO statements in FORTRAN, needs to understand how to simulate these in machine language. Two basic types of loop are presented: the loop starting with INX and ending with CPX and BNE, and the loop ending with DEX and BNE. (Other types will be presented later.)

(12) Offsets (Section 23). This is an often sadly neglected topic. By offsets we mean the use of LDX J followed by LDA T-1,X (rather than LDA T,X), for example, to load T(J) when the first element of the array T is T(1) (that is, when there is no element T(0)); also LDX J followed by LDA T+8,X (for example) to load T(J+8) (or T(J+9) if T starts with T(1), thus combining this with the preceding technique). Here the offsets are respectively -1 and 8. Offsets are indispensable; th s in moving an array U to an array V, with a loop ending in DEX and BNE, it is necessary to use LDA U-1,X and STA V-1,X rather than LDA U,X and STA V,X (since the final value of X is 1, in such a loop).

(13) Character codes (Section 24). Pretty soon the student will want to put messages (like ENTER THE FIRST NUMBER) in programs, and this section, introducing all the basic character code concepts on the APPLE (normal, inverse and blinking mode, double quotes and single quotes, control characters, and so on) is put in to anticipate the student's wishes.

(14) Input-output, subroutines, and EQU (Section 25). Again: pretty soon the student will want to do I/O, which, on the APPLE, is done by means of monitor subroutines. Just the basics are given here: JSR (but not RTS, and nothing about stacks yet); descriptions of each of three common monitor subroutines (RDKEY, COUT, and GETLNZ); and the use of EQU (which is necessary in specifying where these subroutines are in memory). This is a good way to sneak in an introduction to EQU, which causes far more student confusion than one might assume.

(15) BYT and ASC (Section 26). A very careful introduction to BYT is needed, because students frequently tend to confuse BYT with EQU.

(16) The program counter and relative addressing (Section 27). We introduce the program counter relatively late in the syllabus; our feeling is that only now does the student have enough familiarity with assembly language concepts to understand it properly. Relative addressing, which takes some getting used to, is also introduced here (because its definition involves the program counter). It is necessary to understand this in order to be able to hand-translate assembly language to machine language, which is coming up quite soon.

(17) Sign status (section 28). This section is mainly devoted to a careful explanation of why one cannot use the sequence LDA P / CMP Q / BMI ALPHA to test for P < Q, whether P and Q are signed or unsigned. This then leads to an explanation of the use of BCC and BCS for this purpose.

(18) Two-byte operations and shifting (Sections 29-35). Covered here are

two-byte unsigned comparisons, increment, decrement, complement, shifting left and right by one bit, and arrays of two-byte quantities, as well as ordinary 8-bit shifting, multiplication and division by powers of two, and multiplication by ten. Also, the use of shifts and the carry flag for processing of the bits in a byte is discussed.

(19) Table lookup, space-time tradeoff (Sections 36-37). Here we have a philosophical digression: is it more important to save space, or to save time? As usual in such digressions, we aim to show that there is no simple answer, but rather several complex answers. In order to discuss the subject intelligently, we need to know about table lookup, and how to do timing calculations; both of these are explained in detail.

(20) Multiplication, division, input-output conversion (Sections 38-41). Beginning with a discussion of the multiplication and division of general binary numbers, we proceed to the consideration of two very tightly optimized subroutines, one for multiplication and one (already reported in [4]) for division. We then proceed to the use of two conversion routines, one for decimal input and one for decimal output; these will be discussed further later on, since they involve concepts we have not had yet. RTS is also introduced, but only as a return instruction; nothing is said yet about the stack or return addresses.

(21) Running programs on the computer (Sections 42-51). Only now is the student deemed sufficiently prepared to be able to write and run assembly language programs. This complete discussion of the subject includes hand assembly, desk checking, walkthroughs, commands in LISA, stepping, tracing, breakpoint debugging, patching (at the assembly level), and communication between LISA and BASIC. Along the way, there is a thorough discussion of intermixing errors (STA Q followed by Q DFS 1, for example) and overwriting errors (instruction codes being destroyed in the course of running a program).

(22) Logical operations (Sections 52-55). There is one section each on AND, ORA, XOR, and BIT. Several applications of each of these are presented.

(23) Overflow status (Section 56). This is introduced quite late. The reason is that, compared to the other features of the machine, it is not very useful; CMP does not affect overflow, and signed comparisons, which use overflow, are trickier than they seem (a complete discussion of this point is given, with a sample program).

(24) Stacks (Sections 57-64). We may note that data structures, in general, are discussed in a later course; the student, at this stage, has no conception of them, and stacks are thus harder to understand than they might seem. We start this discussion by covering a few topics that our discussion of stacks will illuminate: saved and restored variables, the concept of a return address, and indirect jumps (no other indirect addressing yet). We then introduce stacks, first in the abstract and then with specific reference to the 6502, including PHA, PLA, TSX, TXS, and the actual operations of JSR and RTS, as well as the stack pointer and the page-one stack area. This is followed by a discussion of stack techniques, first in general and then with specific reference to the input-output conversion programs which we mentioned earlier; these are two programs which illustrate, between them, several advanced uses of stacks. Finally there is a discussion of why we use stacks (there are good reasons that have nothing to do with recursion, which is important because most programs do not call themselves, either directly or indirectly).

(25) Decimal mode (Sections 65-66). This includes the decimal mode flag, CLD and SED, and routines to pack strings, unpack strings, and add two packed strings.

(26) The status register (Section 67). This section appears here because we can now treat several of its applications, such as saving decimal mode status in a subroutine (using PHP and PLP) so that the subroutine can use decimal mode, whether its calling program does or not.

(27) Interrupts and input-output (Sections 68-72). The section on the status register naturally leads us into a discussion of interrupt subroutines and why they must save the status register (and the fact that this is done automatically on the 6502). This, in turn, naturally leads us into a discussion of input, output, simultaneous input-output using queues and polling, and the speaker on the APPLE (although we treat only the simplest application here, namely the generation of a musical tone).

(28) Further string declarations in LISA (Section 73), namely INV, BLK, DCI, STR, and HEX.

(29) Page zero and indexed indirect addressing (Sections 74-77). A major decision in this syllabus was to postpone the treatment of page zero almost to the end. It is seldom used on the APPLE by user programs (because using it would overwrite locations used by the monitor, LISA, and BASIC). In turn, we cannot introduce indexed indirect addressing until the student knows about page zero. An important application of post-indexed indirect addressing, namely the processing of arrays having more than 256 elements, is given in a section of its own.

(30) Modification of instruction words (Sections 78-81). Most people shy away from discussing this subject at all. Their feeling is that it is bad programming practice anyway, and hence better

left out of the curriculum. The result, however, is that programmers inevitably discover it for themselves, like it, think they've made a new discovery, and use it in bad ways. We introduce it and cover several applications of it, being careful, along the way, to show the difficulties. Our hope is that programmers either give up on it (which is what many people wanted in the first place), or else learn to use it responsibly. Modification of ordinary addresses, of immediate data, and of relative addresses is covered.

(31) Arrays and sorting (Sections 82-87). This includes arrays of strings, arrays of hexadecimal digits, two-dimensional arrays, and the process of sorting and of searching a sorted array. The connection between sorting alphabetic and numeric data is shown (that is, the interpretation of a sorted alphanumeric array as being "alphabetized" in the usual sense).

(32) Two-byte signed numbers (Section 88). So far we have considered signed and unsigned 8-bit data, and unsigned 16-bit data; here we complete the picture. This also gives us an opportunity to prove rigorously something we have assumed as given -- namely that the same add and subtract instructions work on both signed and unsigned data.

(33) Loops ending in INX and BNE (Section 89). Another type of loop combining advantages of speed and forward processing, but a little more difficult to work with, and with more restrictions.

(34) Tapes and disks (Sections 90-92). This includes a discussion of tape parity checking, the APPLE disk operating system, and a number of further LISA pseudo-operations (DCM, LST, NLS, PAG) and commands (W, control-D), as well as two more monitor subroutines (KEYIN and COUT1).

(35) Simulators, interpreters, assemblers, compilers (Sections 93-95). A general philosophical discussion of these is often given in a first course on BASIC or FORTRAN, but this discussion should be expanded now that the student knows assembly language and machine language.

(36) Structured programming (Section 96). This section is mainly devoted to the connections between structures programming and assembly language, such as how a typical structured programming statement (DO-WHILE, CASE) would be implemented in assembly language.

(37) Real numbers (Sections 97-100). There are no floating point instructions on most microcomputers (even the 16-bit variety), but many students have used floating-point BASIC on the APPLE and are curious about how this is done. We start by discussing binary and hexadecimal fractions, then floating point formats, then floating point operations (including normalization), and finally a discussion of typeless processing (that is, keeping the type of a variable in memory along with its value, and interrogating the type before performing any operations).

The Appendix To The Syllabus

The syllabus is accompanied by an appendix, including the following tables: (1) an introduction to BASIC for those who might know only FORTRAN or PL/I or PASCAL; (2) number base conversion; (3) the 6502 instructions in alphabetical order, together with their meanings and the status flags they set; (4) the 6502 instructions in alphabetical order, together with their addressing modes and machine language forms; (5) the 6502 instructions in the numerical order of their machine language forms; (6) the LISA pseudo-operations and extended mnemonics; (7) all special characters used in LISA, together with their meanings; (8) all addressing modes used in 6502 instructions, together with their meanings; (9) character codes for letters and digits in all modes; (10) character codes for all characters other than letters and digits, in all modes; (11) all the APPLE monitor subroutines used in this syllabus, together with their actions (we actually use only a small number of the available monitor subroutines); (12) all registers and flags and their capabilities (that is, which instructions use them directly); (13) all the LISA commands used in this syllabus, and their meanings; (14) all the APPLE monitor commands used in this syllabus, and their meanings; (15) a table of the various subcodes of the 8-bit operation code of the 6502 and of the "families" of 6502 instructions (depending on the rightmost two bits of the operation code).

We have found that one semester is about right for the coverage of this material. If advanced topics (Sections 49-51, 65-66, 68-73, and 78-100) are omitted, the material can be covered in a quarter, for those schools on the quarter system.

References

1. Maurer, W. D., APPLE Assembly Language, Computer Science Press, Rockville, Md., September 1983.

2. Leventhal, L. A., 6502 Assembly Language Programming, Osborne/McGraw-Hill, Berkeley, Calif., 1979.

3. Osborne, A., An Introduction To Microcomputers, Osborne/McGraw-Hill, Berkeley, Calif., 1976.

4. Maurer, W. D., An improvement upon a division program by Leventhal, Dr. Dobb's Journal 7, 3 (March 1982), pp. 20-21.

* * * * * * * * * *

STUDENT-DOWN SYSTEM DESIGN

Robert Geist

Department of Computer Science
Duke University, Durham, North Carolina

Abstract

A new method for the design of computer systems is put forth, which recognizes that an individual student's perception of system performance may differ radically from that of the system administrators. Recent analytical results from application of the method to the design of dual processor systems are surveyed, and a call is issued to the educational computing community to undertake the empirical research in psychophysics that is necessary to complete a comprehensive methodology.

0. Introduction

One of the most painful problems in educational computing today is the allocation of the scarce (non-existent?) school dollar among the ever increasing multitude of available computer system configurations. Although it is easy to observe that the explosive growth in alternative systems has been accompanied by an equally explosive growth, within the computer science community, of analytic design tools intended to solve the configuration problem (e.g. [1,5,8,11]), it is our contention that such tools are inappropriate for our use in educational computing.

After all, the classical approach to analytic system design (including, until recently, our own [5,6]) has been to tacitly assume a goal of minimizing mean system response time, or mean reciprocal throughput, or system cost, subject to some constraints. Yet, though each of these measures may be held to champion the cause of a particular group (the students, the administration, the bursar), each must be recognized as a purely external measure, not directly available to, and hence not directly measured by, the internally situated users of the system.

Most of us would agree that our students gather system performance measures which differ substantially from those "paper measures" seen by system administrators. The issue is this: can we lend quantification to the measures students gather, and can we use such measures, rather than the classical objective functions, in designing systems? It seems we can.

1. The Perceived Mean

In [7], Harvard's David Hemenway offered an "alternative mean" as the answer to an often-posed student question, "Why are my classes larger than the 'average class size' printed in the school catalogue?" Specifically, if M students populate N classes of sizes $X_1, X_2, \ldots, X_N$, the alternative mean class size is

$$X^* = \sum_{i=1}^{N} (X_i/M)X_i$$

that is, the expected size of a class containing a randomly selected student. Hemenway provides convincing evidence that this measure more accurately reflects the information actually available to the students, who are unable to view all classes from "on high."

In [3], we showed that the natural extension of this alternative mean to the user-perceived mean number of customers in a queueing system turned out to be

$$N^* = \frac{E[N^2]}{E[N]}$$

where N is a random variable denoting the number of customers in the system and E is the ordinary expected value operator. Now in classical analytic system design, we regard the components of a computing system (CPU's, drums, disks, etc.) as servers and jobs as customers, who queue when the desired component is busy. Thus we can represent an entire computing system as a network of queues, and, as might be anticipated, "student-down" design (wherein the student moves to the "top" position) is then that design methodology obtained by replacing E[N] with N^*

wherever the former appears in the classical queueing network approach. Perceived mean response time, R^*, is then given, in accordance with Little's formula (see [6]), by $R^* = N^*/\lambda$, where λ denotes the mean arrival rate of jobs to the system.

The question at hand is now this: if we were to design a system to minimize perceived mean response time, R^*, rather than ordinary mean response time, $E[R] = E[N]/\lambda$, might we reach a substantially different conclusion? Indeed. Even in elementary design problems, the effects of this change are dramatic.

2. A Summary of Preliminary Results

We now present the results from three elementary problems on the design of dual-processor systems. The analytic derivations of these results will eventually appear in [3,4], and will not concern us here. Our aim is merely to give the reader the flavor of the new methodology.

A. Should we have separate queues or a combined queue?

In figure 1 we show two possible configurations for a dual-processor system. The processors service jobs at a mean rate of μ_i jobs/second, i=1,2, and jobs arrive to either system at a mean rate of λ jobs/second, where necessarily $\lambda/2 < \mu_i$, for otherwise the queues grow without bound. For this and all examples, the job service times are assumed to be exponentially distributed and the arrival processes are assumed to be Poisson; these standard assumptions of system design are based on both empirical evidence and analytical tractability.

It is a relatively straightforward argument [3] to show that the mean response time for the separate queue configuration of system 1, $E[R_1]$, is given by

$$E[R_1] = 2/(2\mu_1 - \lambda)$$

and that for the combined queue configuration of system 2 is given by

$$E[R_2] = 4\mu_2/(4\mu_2^2 - \lambda^2)$$

If $\mu_1 = \mu_2$ then clearly $E[R_2] < E[R_1]$, but we usually incur additional management costs associated with system 2, so that $\mu_1 > \mu_2$. The point at which such additional costs would cause us to choose system 1 can be determined quickly from the expressions above:

$$E[R_1] < E[R_2] \text{ iff } \mu_1 > \mu_2 + (\lambda/2)(1-\lambda/(2\mu_2))$$

In particular, if $\mu_1 = \mu_2 + \lambda/2$ then $E[R_1] < E[R_2]$, so we should choose system 1. But from [3], if we should also have $\lambda < (1 + \sqrt{17})\mu_2/4$, then $R_1^* > R_2^*$!

Thus there seems to be a window of parameter values in which customers would prefer the combined queue configuration over the separate queue configuration, even though the ordinary mean response time would not be as good.

B. Should we unplug a weak processor?

Consider again a combined queue, dual-processor configuration, such as system 2 of example A, but now suppose the service rates of the two processors are not necessarily identical. Let them be given by μ_1, and μ_2 where $\mu_1 = \alpha\mu_2$ for some $\alpha \geq 1$.

A classical result from system design [12] states that if we want to minimize mean system response time, then there is some disparity ratio α beyond which we should simply unplug the weaker processor. For example, if $\lambda = .2$ and $\mu_2 = .25$, then for $\mu_1 > 2.825\mu_2$, we should unplug the weaker processor and run jobs solely on the faster one.

Now consider the student-down approach: from [4], the perceived mean response time of the combined system is

$$R_c^* = \frac{1}{\lambda}\left[\frac{(1+\alpha)\mu_2+\lambda}{(1+\alpha)\mu_2-\lambda}\right]$$

whereas that for the more powerful processor standing alone is

$$R_s^* = \frac{1}{\lambda}\left[\frac{\alpha\mu_2+\lambda}{\alpha\mu_2-\lambda}\right]$$

so that $R_c^* < R_s^*$ for all relevant λ, μ_2, and α, and we should always leave the weaker processor plugged in.

C. How should power be allocated between processors?

Consider the system of example B, but now suppose that we are free to choose μ_1 and μ_2 subject to a budgetary constraint of the form $\mu_1 + \mu_2 = K$, a constant. Classical analysis shows [4] that mean system response time is minimized if

we choose

$$\mu_2 = \frac{K}{1 + \sqrt{1+(K/\lambda)}}$$

(and, of course, $\mu_1 = K - \mu_2$). On the other hand, when we incorporate student perception we find (as in example B)

$$R_c^* = \frac{1}{\lambda}\left[\frac{K+\lambda}{K-\lambda}\right]$$

which is independent of the allocation.

So allocation becomes, in effect, a non-problem. Perhaps a more important observation is that this last formulation, together with that for R_s^* in example B, suggests that students perceive any dual-processor system as equivalent to a single processor system having service rate equal to the sum, $\mu_1 + \mu_2$.

3. A Call For Empirical Research

A major factor in the student-down methodology is still missing: R^* involves no psychophysics. We must digress.

If f(a) is a physical scale and g(a) a psychological scale, where "scale" is used in the measurement-theoretic sense (see Roberts [9]) then their relationship ψ:

$$g(a) = \psi(f(a))$$

is called the psychophysical function. In our case, f and g represent system response time measurements, and, up to this point, we have assumed ψ to be the identity, that is to say, we have assumed perfect perception of the information available. The issue raised in the preceding sections is that this available information, and hence any conclusion derived therefrom, depends heavily upon the location of the perceiver, even if his ability to perceive is unimpaired.

To incorporate the ability of the perceiver, we propose to follow the great body of literature from the psychophysics of prothetic continua (of which time duration is one example) and assume the power law, that is, $\psi(x) = \alpha x^{\beta}, \alpha > 0$. From Ekman and Sjöberg [2] on prothetic continua: "As an experimental fact, the power law is established beyond any reasonable doubt, possibly more firmly established than anything else in psychology." Stevens [10] provides evidence that for estimation of time duration of white noise stimuli, $\beta \approx 1.1$.

Now, for a certain collection of problems, neither the precise form of ψ nor that of R^* as a function of N^* is necessary. If we merely wish to compare alternative systems on a basis of perceived mean response time, it suffices to assume that each of ψ and R^* (or, more generally, their composite) is monotone non-decreasing, for system comparisons on a basis of $\psi(R^*(N^*))$ can then clearly be made on a basis of N^* alone.

On the other hand, we contend that the most vital system design imperatives are of the form, "Minimize system cost subject to an upper bound constraint on $\psi(R^*(N^*))$." For such, a precise specification of ψ is mandatory.

To our knowledge, no attempt has been made to establish a power law for time duration in the context of computer system response. The results of any such attempt would be invaluable to our own research, and we call upon the educational computing community to undertake this important study. Only in this way might we be able to exchange our scarce dollars for the maximum student satisfaction.

4. Conclusions

We have proposed a new "student-down" methodology for the configuration design of computer systems. It is our basic thesis that student evaluation of computer system performance differs markedly from administrative evaluation, and that it depends heavily upon the student's location, which is necessarily internal to the system itself. We contend that the perceived mean, originally introduced by Hemenway [7], captures the effects of this location.

Using the new methodology, we often find that the optimal system design differs radically from that obtained through classical analysis. Student-down designs are also usually easier to implement.

The final (and yet unspecified) component of the new methodology, the psychophysical function, is the subject of our call to the educational computing community for extensive experimental research.

This work was supported in part under NASA Langley Research Center Grant #NAG1-70.

5. References

1. Chandy, K., Hogarth, J., and Sauer, C., "Selecting Capacities in Computer

Communication Systems," IEEE Trans. Soft. Eng., SE-3 (1977).

2. Ekman, G., and Sjoberg, L., "Scaling," Ann. Rev. Psychol., 16 (1965).

3. Geist, R., "Perception-Based Configuration Design of Computer Systems," submitted to Information Processing Letters.

4. Geist, R., and Trivedi, K., "The Integration of User Perception in the Heterogeneous M/M/2 Queue," Proc. of PERFORMANCE '83, May, 1983.

5. Geist, R., and Trivedi, K., "Optimal Design of Multilevel Storage Hierarchies," IEEE Trans. on Comp., C-31 (1982).

6. Geist, R., and Trivedi, K., "Queueing Network Models in Computer System Design," Mathematics Magazine, 55 (1982).

7. Hemenway, D., "Why Your Classes are Larger Than 'Average'," Mathematics Magazine, 55 (1982).

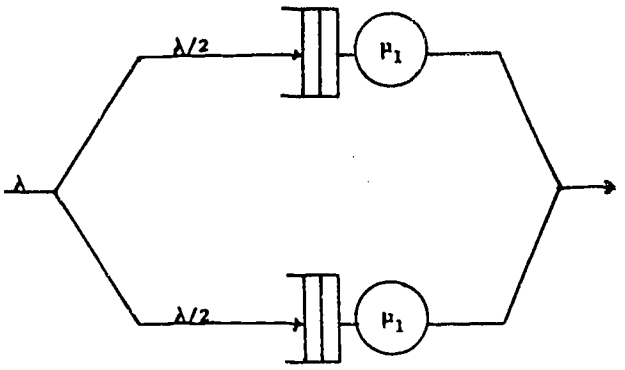
8. Ramamoorthy, C., and Chandy, K., "Optimization of Memory Hierarchies in Multiprogrammed Systems," JACM 17 (1970).

9. Roberts, F., Measurement Theory, Vol. 7, Encyclopedia of Mathematics and Its Applications, Addison-Wesley, 1979.

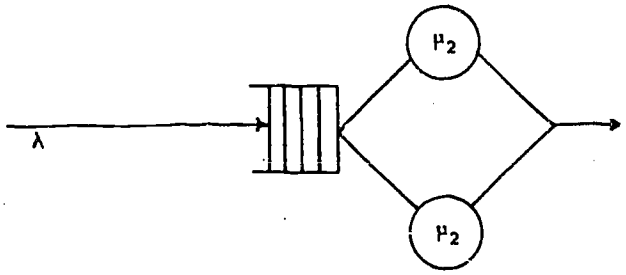
10. Stevens, S., "On the Psychophysical Law," Psychol. Review, 64 (1957).

11. Trivedi, K., Wagner, R., and Sigmon, T., "Optimal Selection of CPU Speed, Device Capacities, and File Assignments," JACM, 27 (1980).

12. Trivedi, K., Probability and Statistics with Reliability, Queueing, and Computer Science Applications, Prentice-Hall, 1982.



System 1, Separate Queues



System 2, Combined Queue

FIGURE 1

COURSEWARE DEVELOPMENT AND EVALUATION

L. Carl Leinbach
Barbara C. Garris
Ann Lathrop
John C. Miller

ABSTRACT: Computing Literacy and the Liberal Arts

L. Carl Leinbach, Chairman, Computing Studies, Gettysburg College, Gettysburg, PA 17325

What is the role of Computer Science in a liberal arts college? This question and Gettysburg College's definition of an answer to the question are the subjects of this presentation.

Gettysburg College is a liberal arts college with an enrollment of 1850 undergraduate students. It is extremely proud of its liberal arts heritage and offers a broad, diversified curriculum consistent with its heritage. In 1976 the faculty published an academic purposes document which clearly defines the role of the curriculum within the liberal arts. That purposes document states that the curriculum must emphasize the following elements:

1. Logical, precise thinking and clear use of language.
2. Broad, diverse subject matter.
3. A rigorous introduction to the assumptions and methods of a representative variety of academic disciplines.

In 1979 the Academic Policy and Program Committee of the faculty conducted a study of the introduction of Computer Science into the curriculum. Previously, computing courses had been taught in the mathematics and business administration departments. The study resulted in the establishment of an interdisciplinary Computer Studies Group which was given two charges:

1. To promote computing literacy on the Gettysburg College campus.
2. To establish a curriculum consisting of not more than four courses for those students who desire to learn more about Computer Science.

The Computer Studies Group developed its curriculum during the 1981-82 academic year and instituted its program in the fall of 1982. The computing literacy course is taught in a new microcomputer laboratory which has 18 Apple II+ microcomputers and 3 Epson MX-100 dot matrix printers. Each microcomputer is equipped with 64K of memory and two disk drives. The course itself emphasizes the algorithmic approach and top-down problem solving. The BASIC programming language is used during the introductory portion of the course, but the main portion of the course is taught using Pascal.

The remainder of the Computer Studies curriculum consists of: Introduction to Algorithms; Data Structures; Design and Analysis of Algorithms; and Computer Organization and Assembly Language Programming. These courses are taught using the Burroughs 6700 computer and the language of choice is ALGOL. The mathematics department supports this program with courses in Discrete Mathematics and Numerical Analysis.

This presentation will focus on the development of the curriculum in view of the Academic Purposes of the College. A special emphasis will be given to the development of the Computing Literacy course, the recruitment of faculty for the program from within the College, and the decisions which led to the establishment of the microcomputer laboratory.

ABSTRACT: Courseware Evaluation Techniques

Barbara C. Garris, Teachers College, Columbia University, Box 27, New York, NY 10027

The Educational Product Information Exchange (EPIE), established in 1966, has been a pioneer in developing evaluation techniques for textbooks and audio-visual materials and equipment which it publishes through a series of subscription publications.

With the advent of a microcomputer hardware and courseware community, EPIE has adapted its evaluation system to the rapidly evolving technology of microcomputers. In partnership with Consumer Union, EPIE hopes to raise the consciousness of today's educators into an informal consumer-oriented Alliance for Quality in Educational Computing, pressuring courseware producers to higher standards than are currently the norm in the courseware market, and to make possible 30-day review copies of software similar to those available for textbooks.

In the short presentation, EPIE will outline (1) how we train groups of teachers to analyze courseware using our own evaluation techniques, and (2) how we evaluate courseware for publication as PRO/FILES Courseware. The Evaluation Coordinator will use slides and transparencies to illustrate the detailed evaluation protocol and follow through the workshop training process which any school system can adapt to its own evaluation and selection of microcomputer curriculum materials. The session will be concluded with a question-and-answer period. Hand-out materials will be provided.

ABSTRACT: The California Courseware Clearinghouse Project

Ann Lathrop, San Mateo County Office of Education, 333 Main Street, Redwood City, CA 94063

The Teacher Education & Computers (TEC) Center program divides California into 15 regions, each with a TEC Center responsible for providing inservice to teachers in the fields of science, mathematics, and computer literacy. The Microcomputer Center in the San Mateo County Office of Education has been designated the software library and evaluation center to provide support services to the 15 TEC Centers for the period from November 1, 1982, through June 30, 1983.

The Clearinghouse will have three major responsiblities:

(1) the training of a cadre of software evaluation specialists who will then train teachers in their regions as software evaluators;
(2) the construction of a three-dimensional subject/computer system/grade level matrix of highly recommended software that will be suggested for preview throughout the state; and
(3) the develoment of subject-oriented collections, each with 10 to 20 promising new software packages, that will be circulated to the TEC Centers on a rotating basis for preview and evaluation.

Successful completion of these three responsibilities will widely expand the base of trained evaluators throughout California, help them to identify the best currently available software for previewing, and also provide interesting new software for evaluation.

Plans are currently underway for a Software Evaluation Institute to be held in San Mateo County in January 1983, for the purpose of developing the matrix of highly recommended software. Representatives from successful software evaluation projects at stae and regional levels will be invited to participate. It is our hope that this group of educators, all experienced in software evaluation at their individual sites, can agree upon the software to be included and help to establish its place in the matrix.

Other activities of the Clearinghouse include the investigation of electronic dissemination of the matrix and of software evaluations, contacts with software publishers, publication and dissemination of software reviews received from the 15 TEC Centers, and the development and distribution of SOFTSWAP public domain programs. There will be a second, smaller Institute held in May 1983 to evaluate the work of the Clearinghouse and to make recommendations for 1983-84.

ABSTRACT: Let's Write Usable Courseware: The City College Algebra Project

Jon C. Miller, 110 Riverside Drive, Apt 14C, New York, NY 10024

The City College of New York has an extensive remedial mathematics program, enrolling almost 1000 students per term in one of two courses which cover basic algebra. In an attempt to provide individualized instruction and student involvement and interaction not attainable in a conventional classroom, the Mathematics Department is turning to computer based instruction.

Existing programs seem not to exploit the full potential of microcomputers in algebra instruction, and the college is therefore developing a set of materials more suited to the students' needs.

Many existing programs are limited to practice problems on a limited range of topics, and are usable only as a supplement to classroom instruction. The City College system covers all the topics needed for a comprehensive course of basic algebra instruction.

Most existing programs do not fully exploit microcomputer graphics capability. The City College system uses graphics in a variety of ways. The Apple II's high resolution graphics capability is used to represent all algebraic expressions in standard algebraic notation. The system represents signed numbers as vectors in order to explicate the rules for signed number operations. It allows exploration of algebraic expressions by a combination of evaluation and graphing.

Most existing programs restrict user input so that, for example, if a fraction is "expected", then only a fraction can be entered. The City College system features a completely general expression entry and display capability, with simple control codes to produce exponents, fractions, radicals, absolute values, and transcendental functions. Virtually any syntactically correct expression can be entered at the keyboard and displayed in standard notation, but a syntactically incorrect character produces an immediate error message.

The City College algebra system allows individualized flow from topic to topic. Excessive errors result in immediate transfer to an appropriate prerequisite topic. After demonstrating mastery of a topic, the student is given a choice of topics to try next, and a choice of whether to start from the beginning or to start with a diagnostic quiz for possible exemption. Assistance is always available by typing a question mark.

Small scale preliminary testing of the earlier portions of this system is in progress, and further portions are being written. The project presentation will include a demonstration of the major features of the City College algebra instructional system and a summary of the preliminary results of students using the system.

Requests for Equipment Proposals

Joseph Wolfsheimer
Division of Automated Services
District of Columbia Public School District
Washington, D.C. 20004

SPONSOR: SIGCAS

ABSTRACT

Numerous technical and procedural issues confront education organizations seeking to procure equipment. The nominal procurement environment becomes more intricate as the quantity of equipment increases. Education's computer related efforts differ in nature from earlier, pilot efforts. They now tend to address large populations of students throughout the educational organization. The need to succeed in providing them computer related experiences places further emphasis for success on the procurement process.

This one and a half hour session presents a case study of several educational organizations which recently procured large quantities of equipment to address multiple educational requirements on a universal basis. Prime emphasis is placed on formulating technical specifications for hardware as an intrinsic portion of instructional planning for computer related efforts.

It is noted that several approaches exist to determining the content of technical specifications. All encompassing specifications are advocated. At the same time, it is held that a generic approach to identifying hardware and software components provides advantages over manufacture-based specifications in cases not involving a large installed base.

An approach to proposal evaluation is suggested. Empirical evaluation is advocated. It is suggested that evaluation techniques be a part of information released to potential vendors in the education organization's request for proposals. A case study is mentioned with a breakdown of advantages and shortcomings.

Finally, approaches to phased procurement are discussed. These include rent, buy, or lease decisions, discount arrangements and open-ended contracts. Budgetary issues are discussed in brief and it is advocated that computational equipment move from capital budgets to some extent.

Courseware on Social Issues of Computers

Ronald E. Anderson, Chair
University of Minnesota

ABSTRACT

New courseware from the Minnesota Educational Computing Consortium and Control Data Corporation offer assistance to those instructors desiring to include computer literacy material dealing with social issues. Such issues include privacy, computer crime, copyright violation, employment, and economic impacts. Software and accompanying text materials for both college and precollege instruction will be demonstrated and discussed. Special attention will be given to techniques for integrating these materials into various types of courses. Consideration will be given to additional needs and future developments.

DISCUSSANTS

Hans Lee
Michigan State University

Beverly Hunter
HumPRO

PANELISTS

Thomas Heaney
Control Data Corporation

Catherine Dunnagan
Control Data Corporation

Richard Pollak
Nancy Kozen
Minnesota Educational Computing Consortium

SPONSOR
SIGCUE

Word Processor in the Composition Classroom

Mary Dee Harris Fosberg
Department of Mathematical Sciences
Loyola University, New Orleans, LA

Donald Ross
Composition Programming English Department
University of Minnesota, Minneapolis, MN

SPONSOR: ACH

PAPERS:

"Writer's Workbench: Teaching Aid and Learning Aid"

Kathleen Kieffer and Charles Smith
Colorado State
Fort Collins, CO

"Aids to Organization"

Helen Schwartz
Oakland University
Rochester, MI

"Studying the Composing Process in the Computer Age"

Lillian Bridwell and Parker Johnson
University of Minnesota
Minneapolis, MN

Interactive Computer Graphics and Computer Animated Films in Education

Maria Mezzina, Chair
Teachers College, Columbia University
New York City, NY

ABSTRACT

The most important issues related to the use of computer graphics in education and the production of computer graphics instructional material at different levels will be presented. The speakers will discuss various applications of computer graphics to education according to their major area of expertise and interest.

Current learning approaches, both with and without the computer, place a heavy emphasis on verbal skills. But it has long been recognized by teachers that not all students have such verbal skills. On the contrary, many students need visual information to aid them in learning. Alfred Bork will illustrate the use of such visual information in computer based learning using examples developed at the Educational Technology Center.

Patricia Harrison will discuss an example of an easy language to be used to design computer graphics software: ZGRASS. Characteristics of the language will be explained. Easy ways to generate graphics and animations for educational purposes will be illustrated visually.

During the period from 1970 through 1977, the Topology Films Project, supported by the National Science Foundation, produced a series of educational films explaining by visual examples various concepts in topology. Nelson Max will show excerpts from the films, and discuss how they were designed, programmed, filmed and edited.

Marial Mezzina will discuss the state of computer graphics in education, giving an overview of systems, languages and experiences. The problem of portability will be addressed. The role of computer graphics in the development of instructional software including conditions for effective use and conditions for easy programming will be addressed by illustrations and examples.

SPEAKERS:

Alfred Bork
University of California
Irvine, CA

Patricia Harrison
Electronic Visualization Laboratory
University of Illinois
Chicago, IL

Nelson Max
Lawrence Livermore National Laboratory
Livermore, CA

Teaching Ada With Computers

George Poonen
Computer * Thought Corporation
Plano, TX 75075

ABSTRACT

Ada is the new programming language that has been adopted as a standard by the Department of Defense. It is expected that Ada will be used for both systems programming as well as large scale real time applications. Consequently there has been a burgeoning interest in the language both in industry as well as academia. In fact, recently, Ada was approved as an ANSI standard.

Ada incorporates many of the important developments in software methodology of the last decade. Many of these features are somewhat foreign even to many of todays experienced programmers. As a result there has been a great deal of concern expressed about education in Ada.

This panel will cover a wide variety of topics ranging from generic issues in transferring complex technology to specific approaches using computers to aid in the teaching of Ada.

PARTICIPANTS:

Lee Blaine
Computer * Thought Corporation
Plano, TX 75075

Peter Wegner
Brown University
Providence, RI 02912

Lt. Col. Vance Mall
AJPO
Arlington, VA 22203

Kenneth Bowles
TeleSoft
San Diego, CA 92121

Electronic Mail and Computer Conferencing

Paul Heller, Chair
EDUCOM/EDUNET

ABSTRACT

This session will describe the electronic mail services and systems that are in use on numerous campuses. A variety of applications will be described. Fundamentally important differences between electronic mail and computer based conferencing will be explained in the context of system features and their effects on patterns of communication within and between groups of participants. Numerous examples will be presented based on use of systems including EIES, COM/PORTACOM, Telemaail, Stanford's CONTACT/EMS, DREAMS, and VAXmail.

A special feature of the session is an introduction to the concept and operation of Mailnet, a service which links local (campus based) electronic mail systems to each other. This service permits persons using their own mail systems to exchange messages and documents with colleagues on other campuses. The Mailnet service has been designed to allow virtually any local mail system to be connected with modest investment of money and technical talent. To design does not require acquisition of new hardware nor any changes to local operating systems. Typical costs for message exchange using the ordinary telephone network are 25 cents per typed page.

PRESENTERS:

Paul Heller
Daniel Oberst
EDUCOM/EDUNET

SPONSOR:
EDUCOM
EDUNET

SEX DIFFERENCES IN MICROCOMPUTER LITERACY

Marlaine Lockheed
Educational Testing Service
Princeton, New Jersey

Antonia Nielsen
Princeton High School
Princeton, New Jersey

Meredith Stone
Educational Testing Service
Princeton, New Jersey

Abstract

In this paper, sex differences in computer literacy, use of computers, attitudes towards computers and motivations toward using computers were examined. Sex differences in observed gain in computer literacy were correlated with computer use, liking and motivation. Statistically significant sex differences were found for both use and liking of computers, but not in expectations for future utility. Liking and motivation were unrelated to gain in computer literacy. Home access to a computer was related to gain for girls only, and after school Computer Center use was related to gain for boys only.

Introduction

> "In all of our sites we observed differential use (of microcomputers) according to sex, particularly at the secondary level. This is not an issue of access per se, since girls are not systematically excluded from using computers. At the elementary level, each sex could and did use the micro more or less equally. But, starting in seventh grade, when the micros moved out of classrooms and hallways into math and business departments, there was an overwhelmingly male representation among students who used the micros" (Sheingold, 1981).

In Tron, the 1982 Walt Disney computer graphics fantasy about computers and computer programming, the hero, his sidekick, and all other major characters, including anthropomorphized programs, the master control program and the Tron program, are male. All characters, that is, except a heroine-programmer whose procedure for laser-beaming matter into electronic impulse is responsible for starting the action, but who plays a relatively minor role in the remainder of the movie. There are no other females in the picture.

In the real world of computers and programming, life mimics art: only 11.2% of doctorates in computer science were awarded to women in 1980-81 (Chronicle of Higher Education, October 6, 1982). Computers are a man's world. But why is this the case? Are females less computer literate than males? Don't females like computers? Don't females recognize that computers will be used in virtually every occupation by the year 2000? What accounts for the numerous anecdotal reports, such as Sheingold's cited above, that girls do not use computers when they are made available in schools? Since computer literacy is emerging as a major educational concern of the present decade (Seidel, Anderson, & Hunter, 1982) the purpose of the study was to answer some of these questions.

Method

During the academic year 1981-82, all students at a suburban, upper middle-class high school in Central New Jersey who were enrolled either in first or second year general mathematics, or in first, second or third year college preparatory mathematics were concurrently enrolled in a required 28-hour computer literacy course that substituted for every sixth mathematics class throughout the year. Students were administered a 15-item test of computer literacy at the first session of the computer literacy course and at the final session; at the penultimate session they were administered a 37-item questionnaire regarding their experience and attitudes towards computers.

1. Measures

Computer literacy pretest and posttest. The same computer literacy instrument was used for both the pretest and posttest. It contained 15 items, evenly divided between general computer knowledge, vocabulary, and programming algorithms. Several items were adapted from the computer literacy test developed by the Minnesota Educational Computing Consortium (Klassen, Anderson, Hansen, & Johnson, 1980).

Computer survey. The survey instrument contained 37 questions regarding student access to and use of computers, student attitudes towards computers, and descriptive information such as grade, sex and mathematics course.

2. Computer Literacy Course

The goals of the computer literacy course were to acquaint students with the potentials and applications of microcomputers, to introduce students to the BASIC Programming language and to give students practice in elementary scientific method. The course was taught in the high school Computer Center by the Computer Center director; the Center itself contained 11 Apple microcomputers with disk drives which students used during class. When classes were not in session--before school for approximately 25 minutes, during the day for two hours and fifteen minutes and after school for forty five minutes--students were also permitted to use the microcomputers. Students were permitted to reserve a computer for programming, but not for games. Most of the students who availed themselves of the opportunity to use the Computer Center during these times played games, although there were some who used out of class time to go beyond the classwork and to complete programs.

The general pattern of the classes consisted of four steps. In each six-day cycle, the students were introduced to new concepts by watching a videotape made by the Computer Center Coordinator which superimposed relevant Apple programs on her video image, by a lecture or by a handout. The introduction to new concepts was followed by a discussion in which student's questions were answered and the classwork was explained. Using the new concepts, the students entered demonstration programs, ran them and made notes on their results. The students were then asked to write programs applying the concepts which had been presented.

Results

Usable data were obtained from 413 (87.1%) of the returned surveys; pretest scores were available for 345 (83.4%) of these respondents and posttest scores were available for 383 (92.5%). Complete pretest, posttest and survey data were available for 114 females and 116 males. Of students with complete records, 74% were ninth or tenth grade students and 97% were enrolled in college preparatory mathematics courses.

1. Sex Differences in Computer Literacy

The first question to be answered addressed the issue of sex differences in computer literacy. From the survey, we found that 62% of the males and 56% of the females had used a computer prior to the computer literacy course, a nonstatistically significant difference. Moreover, no statistically significant difference was found between the mean pretest scores on the computer literacy test of males ($M = 3.49$; $S.D. = 2.21$) and that of females ($M = 3.12$; $S.D. = 1.69$). Even though high proportions of both male and female students had been exposed to computers before entering the course, most students were quite computer illiterate, as we measured computer literacy. The highest score on the computer literacy pretest was 11; it was achieved by one boy (one girl scored 10). The lowest score was zero, which was achieved by seven boys and three girls.

On the posttest a statistically significant ($t = 2.56$; $p < .01$) difference between male ($M = 7.68$; $S.D. = 3.04$) and female ($M = 6.92$; $S.D. = 2.74$) scores was found. The highest score on the posttest was 14, which was achieved by eight boys and one girl. One girl scored zero on the posttest.

To verify that this difference was not due to changes in the student sample between fall and spring, a multiple regression was conducted to assess sex differences in computer literacy gain. In this analysis, which was conducted on data from those respondents having both pretest and posttest scores, the posttest score was the dependent variable and the pretest score was the control variable; gender was considered the independent variable. The results of this analysis are presented in Table 1. For the 318 students having both pretest and posttest data, females gained approximately .75 points (one-fourth of a standard deviation) less than males, a difference that was statistically significant.

2. Sex Differences in Student Computer Practice, Attitudes, and Motivation

The second question to be answered addresses the issue of sex differences in student use of or practice with computers, student attitudes toward computers and student motivation regarding computers.

Practice. Nine questions regarding student use of computers were included on the survey. Statistically significant sex differences were found in the responses to every question but one, regarding prior use of a computer (Table 2). Males reported greater access to computers outside of school than females, greater extracurricular use of the Computer Center, more frequent computer programming and more frequent computer game playing than girls. The largest sex difference was found for reported game playing: 82% of the boys, compared with 48% of the girls, reported having played either a computer game, video games or arcade games at least three times.

Liking. Four questions related to student liking of computers were included on the survey. Statistically significant sex differences were found in the responses to all four questions, with boys reporting more favorable attitudes than girls toward computers and programming. Fewer than half the boys and a third of the girls reported that they liked working with computers or programming, however.

Motivation. Three questions relating to student expectations for future computer use were included on the survey. Sex differences were found on only one of the three questions. Two thirds of the boys compared with less than half the girls reported that they expected to use computers the following year, a difference that was statistically significant. On the other hand, 80% of both boys and girls thought that knowing how to use a computer would be important for them in the future. While few boys or girls reported that they planned to take any other computer courses, there were sex

differences in what types of courses boys and girls planned to take (Table 3). Twice as many boys as girls reported future plans to study programming languages, while more girls than boys reported future plans to study computer applications to business, research or word processing ($X^2(4) = 21.37$, $p < .001$).

3. Factors Related to Computer Literacy Achievement Gain

In general, achievement gain in any subject can be viewed as a function of practice, attitude, and motivation factors, such as those discussed in the previous section. To assess their effects on achievement empirically, we related measures of practice, attitude and motivation to gain in computer literacy.

Practice. The nine questions relating to student use of computers were analyzed separately using analysis of variance with adjusted gain* on the computer literacy test as the dependent variable and the question as the independent variable. Four of the nine practice variables were related to adjusted gain: (1) access to a computer outside of school ($F = 7.314$; $p < .01$); (2) coming to the Computer Center at times other than class ($F = 5.042$; $p < .05$); (3) coming to the Computer Center after school ($F = 4.67$; $p < .05$) and (4) playing computer games, video games or arcade games ($F = 2.778$; $p < .05$). In all cases, students reporting more practice achieved greater gain.

Liking. The four questions relating to student liking of computers were analyzed similarly; none were related to adjusted gain in computer literacy.

Motivation. The three questions relating to student expectation for future computer use were also analyzed through analysis of variance; none were related to adjusted gain in computer literacy.

*Since simple gain scores (the difference between pretest and posttest scores) are relatively unstable, we computed adjusted gain scores (the difference between predicted posttest scores and actual posttest scores) to estimate the significance of these apparent differences in learning. The predicted posttest scores was obtained from an ordinary least squares regression of pretest on posttest, which yielded the following prediction equation:

$$Y' = 0.4610X + 5.926$$

where Y' is the predicted posttest score and X is the pretest score. This equation was applied to each student's pretest score to yield the predicted posttest score. The student's predicted posttest score was then subtracted from his or her actual posttest score to yield an adjusted gain score:

$$G = Y - Y'$$

where G is the adjusted gain, Y is the students actual posttest score and Y' is his or her estimated posttest score.

4. Sex Differences in Factors Related to Computer Literacy Gain

The four practice factors that we found were related to adjusted gain in computer literacy were reexamined for evidence of sex differences (Table 4). In all cases, males reported greater practice with computers than females reported. One third of the males compared to about one-fifth of the females reported having access to computers outside of school. Nearly 40% of the males reported coming to the Computer Center at times other than class, compared to fewer than eight percent of the females, and over 20% of the males reported that they stayed to use the Computer Center after school, compared to fewer than three percent of the girls. Finally, the majority of males (61.3%) played computer games once a week or more, while the majority of females (52.2%) had not played computer games more than twice ever.

5. Sex Differences in Determinants of Computer Literacy Gain

Having identified four practice-related variables that were related to gain in computer literacy and for which sex differences were observed, we examined the relationship between these variables and gain in computer literacy, separately by gender.

In these analyses of variance, adjusted gain was the dependent variable and the four practice variables were the independent variables. Access to a computer outside of school was modestly related to adjusted gain in computer literacy for females ($F = 5.178$; $p < .05$) but not for males. While use of the Computer Center after school was modestly related to achievement gain for males ($F = 4.311$; $p < .05$), it was not related to achievement gain for females. Use of the Computer Center at times other than class (that is, before school or during free periods) and game playing were unrelated to computer literacy gains for either males or females.

Summary

This paper has examined sex differences in computer literacy among secondary school students. Our findings may be summarized as follows:

1. No sex differences in initial levels of computer literacy were found, but boys gained more than girls on computer literacy from pretest to posttest.

2. Males reported more frequent use of computers than females and more positive attitudes towards computers.

3. Although no sex differences in perceived future utility of computers were found, more males than females planned to take a computer course in the future.

4. Access to computers outside of school was significantly related to computer literacy gain for girls, but not for boys.

5. After school use of the Computer Center was significantly related to computer literacy gain for boys but not for girls.

6. Computer game-playing was unrelated to computer literacy gain for both boys and girls.

REFERENCES

Klassen, D. L., Anderson, R. E., Hansen, T. P., & Johnson, D. C. Study of computer use and literacy in science education. St. Paul, MN: Minnesota Educational Computing Consortium, 1980.

Seidel, R. J., Anderson, R. E., & Hunter, B. Computer literacy. New York, NY: Academic Press, 1982.

Scheingold, K. Issues related to the implementation of computer technology in schools: A cross sectional study. Paper presented at the National Institute of Education Conference on Issues Related to the Implementation of Computer Technology in Schools, Washington, DC, 1981.

Table 1

Pretest and Gender Determinants of Posttest Computer Literacy Score for 318 9th, 10th, 11th and 12th Grade Students[a]

	(1)	(2)
Pretest	.460*** (5.778)	.444*** (5.605)
Gender	--	-0.755** (2.441)
Constant	5.926	6.346
R^2	.096	.112
$\hat{R}^2$	.092	.107

Note. [a]The numbers in the table are the unstandardized regression coefficients, with their associated t-statistic below in parentheses.

Table 2

Sex Differences in Use of Computers, Liking of Computers and Expectations for Future Computer Use

	Response Category	Male (N=206)	Female (N=207)	Chi-square
Use of Computers				
Had you ever used any kind of computer before taking this course?	% Yes	61.7	56.0	n.s
Do you have access to a computer outside of school?	% Yes	33.3	22.2	6.14*
Did you come to the Computer Center this year only for class?	% Yes	61.3	92.6	54.14***
Did you come to the Computer Center this year before school?	% Yes	18.0	1.9	27.92***
Did you come to the Computer Center this year after school?	% Yes	21.5	2.4	33.67***
Did you come to the Computer Center this year during your free period(s)?	% Yes	40.8	8.3	56.76***
Have you reserved a computer for programming after school?	% Yes	23.8	15.1	11.01*
How often have you programmed or used a computer for schoolwork outside of class?	% Once or more	28.4	14.1	20.17***
How often have you played computer games, video games or arcade games?	% 3 times or more	81.9	47.8	72.68***
Liking of Computers				
Do you like playing computer games?	% Yes	77.7	63.6	11.94**
Do you like working with computers?	% Yes	49.5	35.0	13.73***
Do you like learning to program?	% Yes	41.5	28.2	19.61***
Do you like writing computer programs to solve problems?	% Yes	27.2	17.6	8.43*
Expectations for Future Computer Use				
Do you plan to take any other computer/programming courses?	% Yes	20.2	14.8	n.s.
Do you think you will use computers next year?	% Yes	68.4	46.3	16.64***
Do you think that knowing how to use a computer will be important for you in the future?	% Yes	81.6	80.2	n.s.

*p < .05
**p < .01
***p < .001

Table 3

Sex Differences in Future Plans for Computer/Programming Courses

	Frequency of Response	
Type of Course	Male	Female
Other programming language	29.3	11.2
Word processing	9.0	19.2
Business/research applications	22.6	32.8
Intermediate or advanced basic	20.1	21.6
Other	9.1	15.2

Table 4

Sex Differences in Factors Related to Computer Literacy Gain

	Frequency of Response		χ^2
	Male	Female	
Access to a computer outside school			
Yes	33.7%	22.2%	6.14*
No	66.3	77.8	
Use of Computer Center only during class			
Yes	61.3	92.6	54.14***
No	38.7	7.4	
Use of Computer Center after school			
Yes	21.5	2.4	33.67***
No	78.5	97.6	
Frequency of playing games			
Never	3.4	5.9	72.68***
Once or twice	14.7	46.3	
About once a month	20.6	23.9	
About once a week	31.4	16.6	
Several times a week	29.9	7.3	

*p < .05
**p < .01
***p < .001

COMPUTERS: LESS APPREHENSION, MORE ENTHUSIASM

Janet Parker
University of Louisville

Constance Widmer
Northern Kentucky University

Abstract

Although the number of microcomputers being purchased and placed in schools is rapidly increasing, many teachers are hesitant, even frightened, by the thought of having to use one. Careful consideration must be given by inservice coordinators about the type of initial computer experiences that should be provided for these teachers, since these experiences can make the difference between acceptance or rejection of computers. This paper will identify possible causes of apprehensions and fears about computers and present specific suggestions for conducting effective inservices to overcome such feelings.

Introduction

The message is clear. It is being broadly stated by public media and professional journals: This is the Computer Age. These wonderful machines have amazing capabilities that can dramatically enhance the way we do our jobs, run our homes, even educate our children. The implication is strong: teachers "should" understand and use computers.

Yet despite all the laudatory publicity given microcomputers, indications are that not all teachers are welcoming them with enthusiasm. Although most teachers acknowledge the importance of "computer literacy" for all students, some teachers outrightly reject using computers; many other teachers are apprehensive, anxious, even panicked at the idea of being expected to use computers. Why do these teachers feel this way? Are their attitudes and fears real? What causes them? What can and should be done to meet the needs of these teachers and, ultimately, their students? Based on the research and experience of the authors, this paper will document the prevalence of computer anxiety among teachers, identify the main fears and misconceptions causing the anxiety, and present specific suggestions for conducting teacher inservices which overcome and alleviate these fears.

Prevalence of Computer Anxiety Among Teachers

In 1976, Lichtman[8] (1979) compared the attitudes towards computers of educators to those of the general public surveyed by Ahl[1] (1976). He found that, "in general, educators seemed less enthusiastic about the computer's role in society than did the general public" (p. 48). The teachers were found to be less sure than the public (64% vs 87%) that computers will improve education; more were convinced that computers dehumanize society (55% vs 37%), and that computers isolate people from social interaction (30% vs 19%). About one third of the teachers reported feeling computers are beyond the understanding of the typical person and more than one fifth did not feel that if one was in their classroom it would help them be a better teacher. Though most (67%) did not feel computers will take their jobs, a disturbing number (16%) did feel this would happen.

Smeltzer[13] (1981) also used Ahl's survey as a basis instrument, but added additional statements appropriate to media specialists. Perhaps reflecting the AV training of the sample, his findings are more encouraging. Only one-fourth of those surveyed felt that computers dehumanize society; most (85%) did not feel that computers are beyond the understanding of the typical person; and most (83%) feel that computers can improve education. In interpreting these encouraging findings, however, it must be kept in mind that the sample size was small (N=29) and select (active members of an educational technology association).

In a much larger, less restrictive study, Stevens[12] (1980) compared the knowledge about and attitudes towards computers of teachers, teacher eudcators and student teachers in Nebraska. Parker and Hazuga[10] (1982) replicated her study with Kentucky educators. Both studies found that although all the educator groups strongly supported the concept that students should be taught about computers, they themselves did not feel qualified to do this teaching. They also found that many educators are uncertain or doubtful that computers can be useful in all instructional areas, and can provide more advantages than disadvantages in the classroom. On items related to computer anxiety, both studies found that as many as one-fourth of the educators were not comfortable around computers, Stevens finding teachers the more anxious subgroup while Parker and Hazuga found it to be the student teachers subgroup, despite the fact that this subgroup had received more training on

computers. As in the Lichtman and Smeltzer studies, both these studies found that as many as 25-30% of the educators are concerned with the dehumanizing aspect of computers, expressing agreement with the statement that, "Computers in education almost always result in less personal treatment of students" (Parker and Hazuga, p. 11).

In addition to such "hard" data, most schools illustrate the phenomena of teacher apprehension towards computers. Some teachers will be "users." They will make use of the computers anytime they can get them into their classrooms. They will come early and stay late to provide extra computer time for their students. They will spend weekends reading computer journals or writing courseware. They will get together with other teachers to share new findings and materials. They will go to computer shows and workshops eager to learn about the latest technological developments. These are the "users". In the same building there will be the "non-users." These teachers will not make use of computers even when available. Why?

As suggested by the surveys cited above, many of them have fears, apprehensions, and/or misconceptions about computers. These feelings are very real to them and effectively act as barriers to their using computers in their classrooms. A first step in overcoming these negative feelings is to identify them and perhaps gain some insight into their causes.

In their work in schools beginning to implement computers, the authors have noticed the following attitudes and feelings among educators. That these feelings are fairly common is suggested by articles expressing similar observations by other inservice coordinators (e.g. Rosell, 1982). What specifically are these feelings?

Fears and Misconceptions

For some teachers, the technology itself of computers is intimidating. They perceive computers as overwhelmingly complex, beyond the understanding of ordinary people like themselves, understandable only to geniuses who use the mysterious jargon and master the intricate circuitry. They are understandably reluctant to use equipment they are convinced they cannot understand. For some, this mind set is antagonized by associating computers with mathematics, toward which they have deep-seated negative feelings.

Also creating anxiety is sensationalist reporting about computers which simulate intelligent behavior, such as the use of anthropomorphic imagery in some recent films. Some envision computers as intelligent machines destined to replace them as teachers. They feel threatened, and fear a loss, or at least a drastic change, in their jobs. They feel a loss of authority and control over their classrooms, over their comfortable, tried-and-true methods, and ultimately over their role in the educational process. They fear that computer companies, rather than teachers, will be handed down to them to be used without question. The computer is seen as a powerful tool for others to exercise control over classroom practice, bypassing the teacher entirely.

Many teachers have become alienated by personal experience with unfriendly programs and systems. They feel computers are sitting there to embarrass them, and they worry about making mistakes and appearing foolish in front of peers and students.

Some teachers associate computers with other technological developments of the past which failed to live up to promises of revolutionizing education. Educational television, teaching machines for programmed learning, fully equiped language labs -- the list goes on. A lot of money was spent, but now the machines collect dust because they never fully succeeded to meet expectations. Teachers see computers as just another of these fads that, too, will pass, costing a lot and producing little learning.

Another fear might be called the "1984 Syndrome", namely that computers will dehumanize education, isolate students, cause them to lose interest in human interaction, and lead to asocial behavior. In the extreme, the vision is of a situation in which people are treated like interchangeable parts of a huge machine.

Effective Inservices

Considering all of these fears which exist in different degrees among a large number of teachers, what can be done? Positive attitudes towards using computers are essential to successful implementation of any school computer-based project. How can negative attitudes and fears be overcome, and enthusiasm toward computer usage be instilled? Listed below are suggestions and caveats for inservice coordinators and teacher-trainers in planning those critical "first exposure" computer workshops for teachers. These suggestions focus on approaches as much as content, for experience has shown that the key to success is to a large extent attitudes: lessen apprehension and increase enthusiasm.

1. Begin the inservice with a guaranteed successful experience. As quickly as possible, have the teachers sit at the computers and run a "friendly" intriguing program with valid educational content. Using one of the Minnesota Educational Computing Consortium* disks with a simulation such as Odell Lake or Oregon Trail has overcome a dozen or more cases of computer phobia. An experience such as this establishes a relaxed attitude. In the process the teachers find they

* Minnesota Educational Computing Consortium (MECC), 2520 Broadway Drive, Saint Paul, MN 55113. "Odell Lake" is on the Elementary Volume 4 disk, "Oregon Trail" on Elementary Volume 6.

themselves can easily control the machine, and that nothing is ruined by pressing the wrong key. They begin to feel in control, finding that they can control the machine and are not being controlled by it. This hands-on experience does much to dispell the mystique a computer holds for many teachers. There will be time later, after confidence is built, to discuss RAM's, ROM's, and CPU's. Avoid technical jargon during this first experience.

2. Have the teachers work in twos or threes at the machines. They are comfortable with asking each other questions, whereas they may be hesitant to ask an unfamiliar instructor. Working in small teams is very effective. Besides helping to catch each other's errors, the team approach lessens the tension which may be experienced when first sitting at a computer. By himself at a computer, an adult may be so tense that he cannot assimilate the simplest directions, and will sit motionless, staring at a screen direction such as, "Press return to continue."

3. If computers are to actually be implemented in classrooms, the teachers must be convinced of their relative advantage over the methods and materials they are durrently comfortable using. The key is to spark their interest by getting them to see the potential of a computer in their classroom. That is, the computer must be perceived as being relevant to high priority items that they wish to achieve with their students, and as being better than what it replaces, not just a nice aid that can be replaced or done without. It must be compatible with the existing curriculum, fitting the established goals, values and needs. One way to show this relevance is to identify a familiar topic the teachers feel is difficult to teach, and illustrate a computer program that can help them teach it. Then actually provide them with the lesson plan and the materials. Ready to go, the computer-based lesson is more likely to be tried, and making that first try is an important step.

4, Also necessary to getting teachers to use and teach with computers is helping them with the classroom management aspects of computer use. The computer will present an atmosphere of discovery and innovation that is unfamiliar to some teachers who use a traditional textbook/lecture approach. Computers will require a certain flexibility, including perhaps a role change in which the student knows more about computers than the teacher. To help teachers with this, provide models of different classroom management systems during the inservice, including total group instruction using one computer with a large monitor, and small group work in a lab mode. Also, during the presentation, do not hesitate to admit it if something happens you, the presentor, cannot explain. It is a rare inservice using computers in which this opportunity does not occur. Admitting we do not know everything, but use computers anyway, does much to build confidence.

5. As documented above, many teachers fear being replaced by computers. Do not avoid this issue; address it directly. After all, the computer was invented by man to relieve man of some of the things he was having to do. In the same way, it can do some of the things teachers are now expected to do. If it could not, there would be justification for having it in the school. Holmes[6] (1982) identifies the heart of the issue with two questions: "First, does the teacher need to perform all the teaching functions?" and, ". . . Can machines perform all, or even most, of the teacher's functions?" (p. 9) To address the first, show the teachers a well done gradebook or other recordkeeping program, and emphasize how the computer can be used to do recordkeeping and other repetitive paper work, freeing them to do the more creative parts of teaching. Then have a frank discussion among the inservice participants listing the kinds of things that teachers do that computers will never be able to do. This will do much to alleviate their fears.

6. In choosing the content for computer inservices, remember that not all teachers need all levels of computer expertise. Do not overwhelm. For example, although most teachers should experience programming in order to understand what a program is and how it controls the computer, they do not need to become proficient programmers. The myth that to use a computer a teacher must be a computer programmer has turned off more than one teacher. The vast majority of teachers will use computers with pre-programmed materials. For them, computer literacy is being able to use the computer with appropriate materials, not being able to write computer programs. This is analogous to most of us with language literacy: we can use books, but most of us do not write them.

7. Also in planning inservices, keep in mind that different people react in different ways to computers. The inservice should provide a variety of activities and approaches. If the session focuses on only one approach, such as programming, only some of the teachers will begin to feel comfortable with computers. At the end of the session, have the participants discuss which of the variety of activities they liked best and why. This will help them understand their own reactions to computers in comparison to others, and will help them when dealing with the various reactions that will surface among their students. As we know, some students will become computer addicts, others will not. Teachers need to be made aware of this.

These are only some of the possible suggestions for implementing effective computer inservices that focus on providing not just the content needed by teachers but also the attitude needed to implement computers in the schools. Yes, we need more funding; yes, we need more quality software; yes, we need more research data on the effects of computers in the teaching/learning process. But these will come. Right now, in the schools, we need less apprehension and more computer enthusiasts.

References

1. Ahl, D. H. Survey of Public Attitudes Toward Computers in Society, in David H. Ahl (Ed.), The Best of Creative Computing, Volume 1. Morristown, New Jersey: Creative Computing Press, 1976, pp. 77-79.

2. Bork, A. Computer Literacy for Teachers. In R. J. Seidel, R. E. Anderson and B. Hunter (Eds.), Computer Literacy: Issues, and Directions for 1985. New York: Academic Press, 1982, 91-98.

3. Bruwelheide, J. H. Teacher Competencies for Microcomputer Use in the Classroom: A Literature Review. Educational Technology, 1982, 22(10), 29-31.

4. Diem, R. A. Developing Computer Education Skill: An Inservice Training Program. Educational Technology, February, 1981, 21, 30-32.

5. Forman, D. Search of the Literature. The Computing Teacher, 1982, 9(5), 37-51.

6. Holmes, G. Computer-Assisted Instruction: A Discussion of Some of the Issues for Would-Be Implementors. Educational Technology, 1982, 22, 7-13.

7. Lazarus, M. So Complicated, They're Simple: A reassuring Word on Classroom Computers. Principal, 1982, 62(2), 39-41.

8. Lichtman, D. Survey of Educator's Attitudes Toward Computers. Creative Computing, 1979, 5, 48-50.

9. Martellaro, H. C. Why Don't They Adopt Us? Creative Computing, 1980, 6(9), 104-105.

10. Parker, J. and Hazuga, M. Educators' Attitudes and Knowledge of Computers in Jefferson County. Unpublished paper, University of Louisville, 1982.

11. Rose, N. R. Barriers to the Use of Educational Technologies and Recommendations to Promote and Increase Their Use. Educational Technology, 1982, 12(12), 12-15.

12. Stevens, D. J. How Educators Perceive Computers in the Classroom. AEDS Journal, Spring, 1980, 221-232.

13. Smeltzer, D. K. The Media Specialist and the Computer: An Analysis of a Profession's Attitude Towards a New Technology. T.H.E. Journal, 1981, 8(1), 50-53.

14. Watt, D. H. Education for Citizenship in a Computer-Based Society. In R. J. Seidel, R. E. Anderson and B. Hunter (Eds.), Computer Literacy: Issues and Directions for 1985. New York: Academic Press, 1982, 53-68.

THE MICROCOMPUTER AS A TOOL IN EDUCATIONAL RESEARCH: A CASE IN POINT*

by Scott W. Brown and Daniel B. Kaye

Department of Educational Psychology, University of Connecticut
Department of Psychology, University of California, Los Angeles

Abstract

Microcomputers have had a major impact in the area of educational research. The present paper discusses some of the major advantages of using microcomputers in cognitive research and presents several examples of our own research programs utilizing the microcomputer. The discussion touches upon the specific hardware configuration, the software, the task demands of the experiments and the transfer and analysis of the data. Conclusions focus on the rationale for the use of microcomputers in educational research discussing such factors as accuracy, reliability and flexibility of the hardware and software.

Introduction

The use of microcomputers for educational and psychological research has become prevalent in recent years. No longer are computers relegated to the statisticians of each research group to crunch and massage data for analysis purposes. With the introduction of smaller and cheaper computers, first the minicomputers and then the micros, many researchers saw an opportunity to use these machines as stimulus presenters and response recorders, in short, as research tools. From a researcher's point of view, what could be more efficient than collecting the data on a computer that may also analyze the data, or at least minimize data loss, by eliminating the procedures for having assistants punch data into the computers for later analyses?

Recently, Johnson has discussed some of the important implications of microcomputers in research. The discussion can be summarized by listing the advantages and disadvantages of this position. The advantages of the use of microcomputers for research are: replication and extension, programmatic research, measurement, experimental control, administration and experimental economy. The disadvantages include the time and the level of training necessary to develop a functional program, and the availability of a number of micros to conduct a study involving a large number of subjects.

With the stage set, we would now like to present a program of research that has been (and still is being) devoted to investigating various types of cognitive processing using microcomputers as research instruments. This discussion will touch upon the hardware and software, program development, experimental administration, data collection, data transfer and statistical analyses relevant to this research program.

The research has been conducted cooperatively at two different sites; the University of Connecticut and the University of California, Los Angeles. Each site has been equipped with the identical microcomputer equipment; thus the same experiments can be conducted simultaneously at both sites without concerns for differences in computer programs.

The purpose of our program of research is to examine the cognitive processes of children and adults, specifically, the acquisition and use of a variety of automatic and controlled information processes related to intellectual development in general and reading skill development in particular. Tasks were developed requiring letter searches within different letter configurations and arrays, lexical decisions (making a word/nonword decision based on the presentation of a letter array), and case judgements (upper case versus lower case) -- typical experimental tasks used by cognitive psychologists, and until very recently involving the use of slide projectors and tachistoscopes.

The microcomputer hardware used in this research program consists of:

1. A North Star Horizon microcomputer equipped with 64K of RAM memory, dual single-sided double density 5 1/4 inch floppy disk drives (180K capacity each);
2. A Televideo 950 black and white video monitor;
3. A Mountain Hardware 100,000 day clock, accurate to within one-tenth of one millisecond;

*Portions of this research were supported by a grant from the University of Connecticut Research Foundation, grant No. 1171-000-22-0401-35-899 awarded to the first author.

4. A Livermore Star modem;
5. An Epson MX-80 F/T printer.

This hardware configuration was selected as the most cost effective system available on the market. In our opinion, the speed, power, storage capacity, expandability and software compatability of the system surpassed its' competition.

The software consisted of North Star Basic CP/M, Pascal/M and two programs to interface the North Star with an IBM for data transfer through the modem (one for the North Star Basic programs and one purchased for the Pascal programs using CP/M). All other software, i.e., task programs and stimulus lists, were written and developed in our labs in North Star Basic or Pascal/M.

The Research

The following is a discussion of four experimental paradigms that have been developed in our labs. These paradigms have been used to investigate reading skill acquisition (experiments 1-3) and the relationship of intelligence to reaction time (experiment 4). These experiments have been conducted successfully with children as young as the second-grade and up through college. All paradigms involved the use of microcomputers to present stimuli and record responses. All forms of counterbalancing, timing, and sequencing of events were programmed and under the control of the North Star.

Experiment #1

The first program that we would like to discuss was used to investigate the effects of contextual facilitation in children and adults. We used a dual task of lexical decision and letter search with words and pseudowords presented on the screen of the Televideo 950. The subjects were required to respond to the letter search or lexical decision by manually pressing one of two predesignated response buttons as rapidly and accurately as possible. The program stored the condition (which of four tasks), the stimuli, the trial number, the reaction time, the subject number and whether the response was correct or not.

This experiment was conducted in two locations. One research site was the laboratory school associated with the University of California, Los Angeles. The other research site was a rural school district near the University of Connecticut. This experiment used students in grades 3 and 6 as subjects. Subjects from each of these grades were included at each research site. In addition, to the data collected on the children, college subjects were tested at each university site. A total of 12 subjects from each grade were included in the study. During the experimental sessions more than 300 reaction times and responses were collected on each subject.

Once the data collection process had been completed for all subjects, the data were transferred to an IBM mainframe computer via a microcomputer interface program and a modem. Upon completion of the transfer, the data were inspected for transmission errors. After the data were "cleaned up" the data were analyzed utilizing a major statistical package on the IBM system. The mainframe was used for the analysis because of the large amount of data and the number of disks involved. In addition, transferring the data to the mainframe freed the microcomputer for continued data collection and program development.

Experiment #2

The procedures for experiment 2 are very similar to those of experiment 1, except for the locations. Data were collected at both Yale University and University of California, Los Angeles. This experiment examined the development of automatic word recognition in children and adults. Subjects were presented with a configuration of words/pseudowords and upper case/lower case. Subjects were requested to make a binary decision regarding different rules for response using these two dimensions of the stimuli. Subjects were requested to make their responses as rapidly and accurately as possible.

Data were collected on both children and adults and transferred to a mainframe for analysis using the same procedures as those described in experiment 1.

The data collected and analyzed from the first two experiments were collated into a study of reading skill development and contextual facilitation in early and skilled readers.[2]

Experiment #3

Experiment 3 was funded under a grant to the first author by the University of Connecticut Research Foundation to investigate several of the components of reading skill development in children[3]. This experiment is actually a combination of three experiments utilizing the equipment described above and a single program. The three experiments involved a microcomputer and a paradigm developed by the first author[3,4]. The paradigm, a primed lexical decision task, requires a microcomputer to present a prime stimulus very rapidly on the screen within a two degree visual angle (the window) of a fixation bar and then present target stimuli in the center of the screen above the bar. The subject is to make a lexical decision (a word/nonword decision) on the target stimuli as rapidly and accurately as possible. The experiments used semantic, phonological or orthographic primes in three separate experiments. The study involved 20 subjects from each of grades 2 through 4 and college subjects for each of the three experimental conditions.

The reaction times, response, stimuli, condition (type of prime) and trial were recorded for each subject. Subjects were presented with 256 trials broken into 8 blocks of 32 trials each. Subjects were encouraged to rest between each block to reduce any fatigue effect. The subjects were able to initiate the task once they had completed the rest period by pressing any button. As an added safety feature, if a subject's fingers drifted off of the response keys and onto any of the

other keys, the bell (built into the keyboard) was sounded to alert the subject and the experimenter. Also, instructions were printed instructing the subject that his/her fingers were no longer on the correct response keys and the subject was requested to realign them. The trial that was presented during this process was scored as incorrect and not included in the statistical analysis. Through these procedures the experiment was under near complete control by the North Star.

All data were transmitted to the IBM mainframe and analyzed using similar procedures to those described previously.

Although this data are still in the process of being analyzed, we are very hopeful that through the use of this specific paradigm and the accuracy provided by the microcomputer and the hardware clock, the subtle patterns in reaction times resulting from the differential use of the primes in processing the targets may be delineated and provide information regarding the development of automatic semantic, phonological and orthographic processing in children.

Experiment #4

This experiment involved the investigation of the relationship between the speed at which a subject may process some very elementary information and scores they may receive on portions of an intelligence test. Recently, this topic has been discussed by Arthur Jensen utilizing fairly simple equipment[5]. Our purpose was to examine some of the hypotheses presented by Jensen and utilize a microcomputer to display and record the stimuli.

The study involved college and elementary school children as subjects and utilized two similar tasks. One task was a simple reaction task in which the subject was instructed to press a single button as soon as he/she saw anything on the screen. The second task required the subject to watch the screen and press one button if an "X" appeared and a different button if an "O" appeared. The reaction time from the onset of the stimulus to the manual response by the subject was recorded for each trial. Each subject was presented with 200 trials of each task.

In order to facilitate recognition of the stimulus when it was presented on the screen, a box was drawn prior to each trial. Subjects were instructed that they were to watch the box because the stimulus was to be presented only within that box, thereby drawing the subject's attention to the proper location. The box occupied the center of the screen and comprised approximately the central 25 percent of the screen.

The stimuli were presented in a predetermined order for the second task because complete randomization would have resulted in an unequal number of X's and O's over the 200 trials. However, a random number generator was used to determine the intertrial duration, since in both tasks anticipation of the stimuli could enhance responses. Through this process each intertrial duration was independent of the others and the effects of anticipation of stimuli was eliminated.

The results of this experiment indicated that as the amount of cognitive processing increases, the relationship between intelligence and reaction time does not increase[6]. Our results seem to indicate that the relationship is fairly constant across these two tasks. Our next experiment in this area will involve the use of a more difficult task involving taxonomic or categorical judgements and will use a hybrid of the program developed for the simple and choice reaction time tasks.

Summary

Four experimental paradigms utilizing microcomputers to present stimuli and record data have been presented and discussed. The advantages of using micros as research tools as discussed by Johnson are certainly true[1]. We have found that through the use of this equipment we have been able to effectively conduct a number of studies within a relatively short amount of time and make a significant contribution to the field of cognitive psychology. Without this equipment we would still have been able to conduct similar studies but the use of this equipment has certainly facilitated the development of our research programs and allowed for collaborative efforts to flourish over 3,000 miles.

As a final note we would like to comment on the two disadvantages of using micros as research tools highlighted by Johnson. The first concerns the amount of time and skill involved in program development. While many of these tasks discussed may appear to be very simple, the initial development whether in Basic or Pascal requires a substantial amount of sophistication in programming as well as the requirements of the tasks, i.e., the types of data required to store, speed of stimulus presentation, placement of targets. This is not to imply that one must be a computer wizard to use a micro as a research tool, but some programming experience is important on a research team. However, the ultimate payoff is worth the effort. For example, with our hardware and software configuration we are able to present stimuli within milliseconds of one another and to record responses accurate to within one millisecond.

The second issue regards the limited number of subjects that may be included in a study if you only have one or very few micros. Both of us have only one North Star at our disposal at each site, yet we each have been able to gather a substantial amount of data from a relatively large number of subjects. The series of three experiments discussed under experiment #3 examined over 320 subjects in grade 2 through college.

In summary, the use of the microcomputer as a research tool may revolutionize educational and psychological research. By using computer programs research may be replicated precisely and extended with minor software modifications. Programmatic research is facilitated through the development of a master program from which hybrid programs may be developed by altering such things as intertrial stimulus duration, sequencing of events, or

instructions. The accuracy of the measurment of such things as reaction time has certainly been increased through the utilization of sophisticated hardware and software. Experimental control is certainly optimized through the use of the micro as a stimulus presenter and data recorder. The use of the micro to administer a task, present identical instructions, and control most aspects of experimental administration is without parallel. Once the computer program has been written the use of the micro is extremely economical. In conclusion, the use of a microcomputer as a research tool can be extremely advantageous to anyone interested in developing programmatic research in a variety of fields.

References

[1]Johnson, C. W. Microcomputer-administered Research: What it Means for Educational Researchers, Educational Researcher, 1982, 11, 3, 12-16.

[2]Kaye, D. B. & Brown, S. W. Contextual Facilitation in Early and Skilled Readers. Submitted for publication, 1983.

[3]Brown, S. W. Reading Skill Development: A Question of Component Processing. A Research Grant submitted and funded by the University of Connecticut Research Foundation, 1981.

[4]Brown, S. W. A Developmental Examination of a Primed Lexical Decision Task, Unpublished Doctoral Dissertation, Syracuse University, 1980.

[5]Jensen, A. 'g': Outmoded Theory or An Unconquered Frontier? Invited address presented at the Annual Convention of the American Psychological Association, Toronto, Ontario, Canada, 1978.

[6]Brown, S. W. & Boretz, H. Cognitive Processing and the Speed of Processing Information in Simple and Choice Reaction Tasks. Paper presented to the National Association of School Psychologists Conference, 1983.

STRATEGIC CONCERNS IN ESTABLISHING AN ELEMENTARY SCHOOL MICROCOMPUTER INSTRUCTIONAL SYSTEM

by Ronald Bearwald and Theodore Bargmann

School District 74
Lincolnwood, Illinois

Abstract

The microcomputer has become a most viable instructional device. There can be no doubt that it can enhance and facilitate learning. However, we dare not lose sight of our educational purposes as we become enamored with the captivating characteristics of the microcomputer. Planning and foresight are essential to insuring the successful contribution it can make to education. This article highlights the strategic concerns of prior planning that lead to effective use of the microcomputer in an elementary school setting. (1) Adopt a developmental approach. (2) Have a broad base of participation in planning. (3) Assess and solicit Board support early. (4) Activate community support. (5) Build your program on a curriculum. (6) Begin with computer literacy. (7) Designate adequate personnel. (8) Provide for a broad base of student participation. (9) Establish a multilingual program.

It is clear that a vast number of private and public, elementary and secondary schools throughout the country are currently investigating or using microcomputers. A recent study conducted by the U.S. Department of Education's National Center for Education Statistics indicates that the number of microcomputers used for instruction in public schools has tripled between 1980 and spring of 1982. The amount of microcomputers now in schools, which is approximately 96,000, is expected continue to grow rapidly.[1]

To be sure, technology has assumed an increasingly more legitimate role in teaching and learning during the past two decades. Taken in this context, the application of microcomputers to education is perhaps an understandable phenomenon. In fact, never before has new technology created such a rapid and profound impact upon the educational community. Never before has an instructional innovation received such widespread support from teachers, administrators, students, and parents. Never before have these groups been so unanimous in their resounding surety of the value of a technological system to the educative process. Perhaps the current sentiment is best captured in the words of renowned psysicist and computer science educator Alfred Bork: "At the present time we see the formation of another great wave that involves the use of the computer in learning. The computer promises to revolutionize education more than any other development of our time."[2]

As one assesses the value of microcomputer-based instruction, a healthy perspective is to view the vast potential and promise as well as the possibility of failure. The enormity of the immediate impact of the microcomputer can be explained in part by its many intrinsic qualities. It has the capacity to provide immediate reinforcement of response even beyond wildest Skinnerian expectations. It can allow a relatively naive learner to perform complex learning operations with a few simple commands. Furthermore, it uses its memory of pre-programmed information to engage any student in logical reasoning regardless of ability. Finally, there is no drawback in that it has the capacity to generally be colorful, noisy and just plain fun. These and other inherent qualities have caused us to become immediately enamored with and receptive to the potential of the microcomputer.

Despite many characteristics which have encouraged many to jump on the microcomputer "band wagon," caution should be exercised. Remember, afterall, that microcomputers are only a delivery system and even the most rudimentary student of the art is familiar with the GIGO, garbage-in, garbage out credo. As has been noted, the microcomputer can be viewed as a "double-edged sword which can cut both ways. While improving the human condition, the microcomputer can create problems and mistakes."[3] Without appropriate planning and/or support, the value of microcomputers may, indeed, be illusionary and transitory. To coin and at the same time reverse a familiar phrase, without a thorough and thoughtful approach microcomputers used in education could become the "silk purse" that turns into a "sow's ear."

At this time, the Lincolnwood School system is operating a successful microcomputer instructional program. The following were key considerations in establishing this program.

ADOPT A DEVELOPMENTAL APPROACH

That is to say, allow for growth of your

program in stages or phases. The value of viewing the innovative process in this manner was clearly stated by Everett Rogers who described stages in the adoption of innovations as (1) awareness, (2) interest, (3) evaluation, (4) trial, and (5) adoption.[4] This approach provides two important aspects of developing a successful program foundation:

1. A conscious recognition which realizes that the gestation period of an effective microcomputer program is months if not years. This view should prompt an appropriate, long-term commitment from those involved.

2. Phases or stages which can be assessed and enjoyed as milestones of success along the way. This in itself can be a motivating force since it provides a basis for progress that is both incremental and continuous.

Our experience has shown these phases to be essential:

PLANNING

This phase allows for the gathering, analysis, and synthesis of data from a variety of sources such as readings, consultants, teachers, and the community. Site visitations to assess on-going program are also a valuable source of information. The planning not only provides a format for gathering information and formulating goals, but also serves as an in-service training component for the planning body.

PROPOSING

During this phase we should seek to inform and motivate the ultimate decision making authority, which in educational settings is often the Board of Education. Broad program parameters should be clarified and feedback solicited.

INITIATING

This is equivalent to a trial stage wherein the innovation is presented and used on a small scale to determine its usefulness when the program is fully implemented. In our situation, this involved providing a structured in-service orientation program as well as placing single microcomputer systems in each school building for teacher and selected student use. Testing a prototype and engaging in preliminary evaluation has proven to be invaluable to introducing microcomputers into the educational environment.[5]

IMPLEMENTING

This stage places your program into full operation. Teaching and hopefully learning begins. Of course, this assumes that these prerequisite tasks have concluded: (1) Board of Education approval of your plan, (2) hardware and software purchased and in place, (3) appropriate personnel appointed, (4) curriculum developed, (5) students grouped and scheduling arranged, (6) professional staff oriented and receptive.

ASSESSING

This phase is on-going and absolutely essential to make your program a "system." Your efforts here will provide valuable data for continuation, expansion or re-direction of your use of microcomputers. Care should be exercised to develop a model which will assess both the cognitive and affective impact upon students as well as the attitudes of faculty.

HAVE A BROAD BASE OF PARTICIPATION IN PLANNING

Whenever possible, use the committee approach which enables a high degree of teacher involvement. It should be taken as good advice that "we must rely very heavily on the intuition of good teachers" in order to determine how "to best proceed and as to where the computer will be most effective in the learning process."[6] In assembling a committee, insure that representation includes broad general areas (reading, math, science) as well as more specialized teaching operations (gifted, library/learning resources, special education). It is also advantageous to include the sophisticated as well as the naive in terms of each person's prerequisite knowledge of computers. As an example, our committee consisted of the following representation:

- administration
- mathematics
- science
- primary teaching (grades K-2)
- intermediate teaching (grades 3-5)
- upper grade teaching (grades 6-8)
- library/learning resources
- reading
- gifted education
- special education

In providing for such broad-based representation, you encourage dialogue and broaden perspective about possible applications. At the same time, you are establishing a cadre of educators that have initial ownership and investment in the program and will, no doubt, provide a base of on-going and future support.

ASSESS AND SOLICIT BOARD SUPPORT EARLY.

There's no need to keep any secrets from your Board of Education which will ultimately

take action that will either launch or doom your intended program. Don't wait for THE FINAL REPORT OF THE MICROCOMPUTER COMMITTEE. Keep them informed along the way by making interim reports and encouraging dialogue. "Yes, we believe microcomputers have potential for our instructional program." "Yes, we are currently investigating this in a systematic manner." "Yes, we will be asking you to support the impending microcomputer instructional program with both hardware and software purchases."

By approaching information sharing in a straightforward manner as this, expectations and support are clear while hidden agendas are kept to a minimum.

ACTIVATE COMMUNITY SUPPORT.

An important compliment to Board interest is that of community support. In attempting to amalgamate support from the citizenry it is essential to consider both the focus and timing of your efforts. In targeting one's energies, one must realize that community support may come from a number of sources:

- parents with children now in your schools.
- parents whose children once did but no longer attend your schools.
- residents who do not or never have had children in your schools.
- former students now in high school or college.
- other public institutions or bodies who are directly or indirectly connected with your schools (parent/teacher associations, public libraries, etc.)
- civic organizations (Chamber of Commerce, Rotary, etc.)
- the business community.

All of these "publics" would have a different vested interest in either supporting or not supporting your microcomputer program. Of course, parents of children currently in your schools will be the greatest single source of support for your program once they realize the potential of microcomputers to increase their children's learning. However, it is important to remember that parents of school children are often far outnumbered by residents who have no direct connection with schools. As an example, in our community, households without children in attendance at school currently outnumber homes with school children by a five-to-one margin. Under these circumstances, it becomes clear that the tendency of the community in general to support a microcomputer program increases proportionately as citizens perceive uses for the microcomputer beyond the school itself. During initial phases, the school must provide for community input in planning as well as for participation in orientation. As the initiation and implementation phases begin, opportunities which personalize microcomputer use must be sought by offering evening and weekend workshops as well as instances to utilize the microcomputers on an individual basis.

The same opportunities for involvement in both planning and utilization should be directed at other public bodies and/or institutions and civic organizations. Library boards, school boards, village boards, zoning boards, chambers of commerce, etc. often provide an interfaced networking of people whose support is crucial. When you can work in concert with these groups to establish common goals and utilization patterns, your efforts can be complimentary if not advantageous.

In our situation, we were able to obtain financial support from our local business community by demonstrating a viable instructional plan for using microcomputers. This type of enterprise has been effective in numerous situations throughout the country. However, willingness of the community to contribute seems to hinge on their awareness of the relevance of microcomputers to the curriculum.[7]

BUILD YOUR PROGRAM ON A CURRICULUM.

The ultimate success of your program will hinge upon how educationally sound it is and/or whether or not learning occurs. Many microcomputer programs are often doomed to failure from the start as a result of not being firmly based on a curricular plan. We must never forget that "the major problem with computer-based learning is not a hardware problem, but a learning problem. To focus on the hardware is an error, drawing attention away from the major issues that should be considered."[8]

Initially, when the microcomputer hardware is in place, the tendency is often to use commercially prepared software as the main framework of the instructional program. This approach can be dangerous on a number of counts. First, there is a substantial belief that the majority of electronic courseware is "intellectually bankrupt" and, indeed, "mental chewing gum instead of protein."[9] Using commercially prepared software as an expedient way to get people to use computers is at best a weak rationale. In doing so, we merely allow ourselves to continually sidestep crucial curricular issues by adopting selected software in a wholesale manner without judging its value of applicability to our situation. In fact, evaluation and selection of microcomputer software should always be done with a maximum of teacher involvement who base their decisions on an accepted set of criteria and an established curriculum.[10]

To be successful any instructional system using microcomputers must be based upon a curriculum - a sequenced set of learning objectives and activities. To be sure, the curriculum document provides a plan for learning. However, it also addresses cost effectiveness and serves as a strategic management tool.[11] Microcomputers obtained with a considerable expenditure of capital outlay should not lie dormant nor be

subjected to capricious and haphazard use. While some random and exploratory use by students should be allowed, the core instructional program should be constructed in order to maximize microcomputer use while insuring that every minute of student participation is productive.

After expending considerable money to purchase microcomputers, it makes little sense to be parsimonious in establishing a curriculum. While curriculum development may be initially time consuming and even costly, it will not only provide direction but a necessary framework for assessing the success of microcomputer-based instruction.

BEGIN WITH COMPUTER LITERACY.

Essentially there are two broad uses for microcomputers in an educational setting:

1. to supplant or supplement instruction through application of computer-managed or computer-assisted instruction.

2. to teach the elements of computer literacy including programming.

Both uses, of course, have merit so it is not suggested that deciding how to apply microcomputers be looked upon as an either/or issue. However, when beginning, it is important to narrow the focus of one's energies in order to enhance the possibility of success. Ultimately, the question becomes not "which approach?" but "which approach first?"

It is, of course, easier to begin a program by relying upon a CAI (computer-assisted instruction) approach using prepared software as its basis. Often this results in purchasing and "plugging in" commercially prepared software either adopting them in total or adapting them as needed. The idea of using commercially prepared software is not in itself abhorant. However, we must never lose sight of the fact that CAI should never be an end, but merely a means to an end. The most legitimate rationale for adopting a CAI approach should be to improve learning within the context of the curriculum. Consequently, such an approach should be implemented only after considering such key questions as "In what way can microcomputers teach something better than we are currently doing it?" In this manner, care will be exercised in integrating CAI into the accepted framework of curriculum.

In our judgment, initiating a program on the basis of a sequential curriculum of computer literacy skills provides a viable beginning as well as a lasting foundation. The reasons are numerous. Using microcomputers to teach computer literacy skills:

1. forces us to specify our goals and activities and deal with (rather than avoid) other important curricular issues.

2. provides a basis for uniformity and continuity of instruction.

3. insures mastery of basic computer and reasoning skills which have universal application to all areas of learning.

4. presents an instructional program which is accountable to varying skills of students.

5. prompts teacher orientation, acceptance, involvement and interest.

Definitions of computer literacy remain varied. Some choose to think of it in terms of two major components: (1) computer awareness and (2) computer programming.[12] A consensus review seems to include these common components: (1) how computers are used, (2) what a computer can and cannot do, (3) what a program can and cannot do, (4) how computers work, (5) how to use a computer, (6) the impact of computers on society, (7) how computers can develop skills of decision making and coping with change and, (8) an introduction to or awareness of programming.[13] The Lincolnwood Schools have adopted the following broad curriculum goal: "To develop computer literacy by teaching important computer related concepts, increasing awareness of the values and applications of computers in our world, and providing opportunities to attain a certain level of competency in performing fundamental computer operations."

Regardless of the operational definition of computer literacy to which you adhere, one could not disagree with the position of Daniel Watt, Director of the Computer Resource Center, Cambridge, Massachusetts who argues the importance of computer literacy. States Dr. Watt, "Universal computer literacy is a basic skill of the 1980's and deserves a major role in the school curriculum."[14] Maxine Greene of the Teachers College, Columbia University, sees literacy as a way "to learn how to think conceptually, to structure experience, to look through wider and more diverse perspectives at the lived world."[15] Furthermore, Dr. Greene reminds teachers that "literacy ought to be conceived as an opening, a becoming, never a fixed end."[16]

Perhaps this is the most useful perspective of all. For, if teachers can grasp the universality of computer skills to learning without feeling they are about to be replaced by a machine, they are more likely to engage in training, evaluation of software, development of courseware and curricular integration of software - all a prerequisite to success in any computer-managed or assisted approach. When one views literacy, as Dr. Greene does, in terms of an

"opening," the case for beginning with computer literacy becomes quite clear. After all, what better place to begin than at the beginning.

DESIGNATE ADEQUATE PERSONNEL.

In considering what resources are necessary to implementing a microcomputer program, school systems must never underestimate the value of providing adequate human resources. Serious consideration should be given to the appointment of an individual who can coordinate the microcomputer instructional program on a full-time basis. Someone must provide continuous impetus to the program, especially during its early stages and to focus efforts to integrate microcomputers into the educational setting.

In addition, this person will need to be a trainer, curriculum developer and teacher. All of these activities will not only benefit the program, but also will enable this person to remain in touch with the substantive issues of microcomputer learning thereby enhancing his/her credibility in the coordinator role. It goes without saying that this person will also need to be a program advocate, ombudsman, promoter and overall champion of the microcomputer cause.

When selecting a coordinator, emphasis should be given to appointing an educator rather than a technician or programmer. Someone who approaches the task with a clear understanding of the educative process will not only enable your learning objectives to be achieved, but will also have a better chance to be perceived by colleagues as a credible leader and therefore accepted. It is even more desirable to designate, whenever possible, someone who is currently employed in the local school system as their credentials as an educator will have been established.

PROVIDE FOR A BROAD BASE OF STUDENT PARTICIPATION.

The tendency of many schools has often been to approach the initiation of microcomputer instruction in a piecemeal fashion. The idea is often to get a "foot in the door" by obtaining one or two microcomputers and place them where it can be presumed they will be accepted and successfully applied. By building a showcase program with gifted students or perhaps in math or science, it is hoped that we can trigger a catalytic reaction from which the program will grow and expand. This premise as a beginning is really not all that bad. The problem is that many programs never get beyond this point. What is thought of as a beginning becomes perpetuated as a series of unrelated and often isolated programs with no common thread of continuity or curriculum holding them together or providing direction.

If a microcomputer instructional program is to be successful over time, it must be committed to provide learning for all students. Assuming that microcomputers are worthy of incorporation into the educational program, then all students should have access to them and the related skills that they teach. After all, if acquisition of computer literacy has the capacity to improve learning in all areas, then access to this skill improvement should not be exclusive. Approaching the initiation of such a program as based upon broad student involvement serves to underscore the magnitude of one's commitment to the belief that microcomputer literacy skills are a valuable component of the curriculum and important to each student's learning.

In our situation, once we agreed that the teaching of computer literacy had a legitimate place in our curriculum, we proposed to provide relevant learning experiences to heterogeneous groups of students at the primary, intermediate and upper grade levels. In doing so we hope to:

1. encourage and obtain a serious initial commitment from the local Board of Education rather than expanding the program with a series of add-on proposals.

2. provide computer skills access to all students thereby engendering a widespread feeling of ownership and involvement by all students and subsequently their teachers and parents.

3. create a substantial hardware base and mainstream instructional program from which additional specialized programs and microcomputer uses would derive and evolve.

ESTABLISH A MULTILINGUAL INSTRUCTIONAL PLAN.

By this we mean, make a serious commitment to expose students to more than one computer language. Commonly, students have been introduced to microcomputer programming through the BASIC language. Though this may be the most universally used language throughout elementary schools, it may not be the most appropriate for the young learner.

Before we proceed perhaps it would be advisable to reiterate our belief regarding the importance of programming experiences at the elementary level. To be sure, elementary students should engage in programming as part of their computer literacy instruction. Too much is to be gained to ignore the potential of this type of learning. As indicated by Seymour Papert,"The child programs the computer and in doing so, both acquires a sense of mastery over a piece of the most modern and powerful technology and establishes an intimate contact with some of the deepest ideas from science, mathematics and the art of intellectual model building."[17]

Furthermore, Alfred Bork notes that when we allow students to engage in programming, "we are giving the student an important intellectual tool, an increasingly critical mode

for all areas of the future."[18]

After reaching consensus regarding our intent to include programming in the computer literacy unit, we determined there would be a great deal of merit in providing a developmental approach which would introduce students to more than one computer language. Logo, for example, and its "turtle" graphics component is highly motivating, does much to eliminate syntax errors and allows primary age students to engage in programming with a minimum of training.[19] The use of such child-appropriate languages as Logo, can enable children to learn to control a microcomputer in the same manner that they learn to read or write.[20] Users of the programming language PILOT have praised its simplicity and clarity which allows the student with no prior computer experience to interact with the machine in a human way. It provides an alternative to algebraic languages and is composed of powerful and nearly syntax-free conversation-processing instructions.[21] We chose to couple these two languages with the widely used BASIC to create the following microcomputer language sequence used in our computer literacy instruction:

Grade Level	Computer Language
Primary (1-2)	Introduce: Logo
Intermediate (3-5)	Review: Logo
	Introduce: PILOT
Upper Elementary	Review: PILOT
(6-8)	Introduce: BASIC

CONCLUSION

The application of both formal and informal procedures to effect change will, of course, vary in accordance with the attitudes and resources present in the situation. Each school or school system must approach the introduction and adoption of new technology in a manner which is suitable to the local educational setting. The introduction of microcomputers for learning may require that we give attention to a unique set of considerations. We have attempted to focus issues which were important to the integration of microcomputers into the educational program of the Lincolnwood Schools. The relevance of any innovative procedures which you choose will be tested in light of the impact that microcomputers make upon the system at large - in fact, upon learning.

"A computer system does not exist in a vacuum. It is always a part of a larger human 'system'. . .

. . . . . A computer system should never be an end in itself. Its success or failure is measured by the success it effects in the situation in which it is used. Therefore, if computer systems are to serve people's needs, they must be carefully integrated into the human and procedural domain they are intended to improve."[22]

REFERENCES

1. "CPR News Briefs," Curriculum Product Review, December 1982, pp. 6-7.
2. Alfred Bork, Learning With Computers (Bedford, Mass.: Dijital Press, 1981), p.3.
3. H. Dominic Covvey and Neil H. McAlister, Computer Consciousness: Surviving the Automated 80s (Menlo Park, California: Addison-Wesley, 1980), p. 6.
4. Everett M. Rogers, Diffusion of Innovations (New York: The Free Press, 1962), p. 81.
5. William H. Pritchard, "Introducing Instructional Computing into the Educational Environment," Electronic Learning, September 1981, p. 24.
6. Bork, p. 7.
7. Robert Neumann, "Doing Business with Business: How to Raise Money in Your Community," Electronic Learning, September 1982, p. 43.
8. Alfred Bork, Learning-Not Hardware-is the Issue," Electronic Learning, September 1982, p. 13.
9. Epiegram, May 1982, p. 3
10. J. D. Gawronski and Charlene E. West, "Computer Literacy," ASCD Curriculum Update, October 1982, p. 6.
11. Fenwick W. English and Betty E. Steffey, "Curriculum as a Strategic Management Tool," Educational Leadership, p. 277.
12. Gary G. Bitter, "The Road to Computer Literacy: A Scope and Sequence Model," Electronic Learning, September 1982, p. 60.
13. Gawronski and West, p. 3.
14. Daniel H. Watt, "Computer Literacy: What should schools do about it?" Instructor, October 1981, p. 87.
15. Maxine Greene, "Literacy for What?" Phi Delta Kappan, January 1982, p. 329
16. Greene, p. 326.
17. Seymour Papert, Mindstorms (New York: Basic Books, 1980)
18. Bork, p. 6.
19. Molly Watt, "What is Logo?" Creative Computing, October 1982, pp. 112-113.
20. Watt, p. 86.
21. Rita May Liff and Keith Vann, "PILOT: A Programming Language for Beginners," Interface Age, September 1978, pp. 64-67.
22. Covvey and McAlister, p. 12.

EVALUATION OF MICROCOMPUTER SOFTWARE: HOW VALID ARE THE CRITERIA AND PROCEDURES?

Robert M. Caldwell, Ph.D.

The University of Texas Health Science Center at Dallas
Department of Allied Health Education
Dallas, Texas 75235

This paper presents three major issues related to the evaluation of educational software. The first relates to the nature of criteria used to evaluate software. Few of the criteria used to evaluate software have been validated through research and experimentation; instead, they often find their basis in speculation and intuition only. Criteria for evaluation are frequently highly inferential in nature which, of course, makes them highly subjective. Subsequently, subjectivity can only serve to lower reliability among separate ratings.

A second problem related to the evaluation of software is the qualifications of the individual raters themselves. Recent studies have demonstrated wide variances among ratings by individual raters.

A third problem in evaluating software stems from a general lack of knowledge about how computers should be used in instruction. Many school districts are acquiring micros without first designing classroom models for instruction. Therefore, much software is purchased without regard to how its objectives fit into existing curricula and the overall goals of the school.

INTRODUCTION

In the past several years there has been a proliferation of software developed to be used with microcomputers in elementary and secondary schools. Unfortunately, much of this software varies greatly in its quality and scope. In an effort to distinguish quality software from "overnight" or "instant" software programs, journals devoted to this new field of "electronic education" have initiated columns where software is reviewed. These reviews typically describe the purpose of the software, summarize its content, and usually review its perceived effectiveness as a teaching medium. These reviews can serve an important purpose in helping set standards of quality in software development but can also have the deleterious effect of condeming a product that might be useful in certain contexts.

If the product is terrible, then it certainly deserves criticism. Terrible software is usually easy to evaluate. It contains inaccuracies in spelling, content, and structure; is a victim of ineffective lesson design, and suffers from poor screen display and appearance. This type of software, however, represents only a small portion of all the programs available. The major problem with evaluating software programs beyond those which are clearly "terrible" is that it becomes difficult to discriminate between the effective and ineffective ones. Programs that are loaded with graphics and flashy screen displays might not be as effective as those that are quite simple in appearance and presentation.

In short, the problems inherent in evaluating educational software can be traced to three major issues. The first relates to the nature of the criteria used to evaluate software. Few of the criteria used to evaluate software have been validated through research and experimentation; instead, they often have their basis in speculation and intuition only. Criteria for evaluation are frequently highly inferential in nature which, of course, makes them highly subjective. Subsequently, subjectivity can only serve to lower reliability among separate ratings.

A second problem related to the evaluation of software is the qualifications of the individual raters themselves. In a recent study by Blum (1982) wide variance was found in ratings of software by three or more reviewers. This variance was attributed to

A. Error in scoring due to inadeuqate training or background of the reviewers.
B. Subjective judgments of the evaluators on items that were highly inferential.

A third problem in evaluating software stems from a general lack of knowledge about how computers should be used in instruction. The emergence of the inexpensive microcomputer has prompted many school districts to purchase them. Unfortunately, many of these districts are acquiring these micros without first designing classroom models for instruction. The result is that much of the software is used and therefore reviewed within contexts that might be inappropriate. In addition, most software is purchased without regard to how its objectivities fit into existing curricula and the overall goals of the school.

These issues, therefore, seem critical to the establishment of valid criteria and procedures for determining effective use of computer-based instruction. In the following paper each of the issues cited above is discussed and recommendations for developing more effective criteria and methods are offered.

Evaluating Evaluation Guidelines

As more software programs become available, the number of individuals and organizations who recommend guidelines for evaluating software seems to increase proportionately. As Chairman of the National Council of Teachers of English Committee on Instructional Technology, I have been currently involved in developing a set of criteria for evaluating computer-based materials for the English language arts. In the process, the Committee has examined at least ten (10) separate sets of criteria. In all these sets of criteria,

three basic problems seem to emerge. The first problem is that many of the criteria on the evaluation instruments are presented as if they are absolute indicators of program effectiveness when in fact they are based more on conjecture and intuition than on evidence of instructional effectiveness. Few, if any, of these criteria offer any support to show that they are the result of a research effort to establish a relationship between a specific feature of a program and its effect on the achievement of the learner.

Another problem with the criteria we examined was that they invariably represent each separate criterion as having equal weight in the evaluation of a particular piece of software. None of the guidelines seem to acknowledge that certain features of program design might contribute more to the overall effectiveness of the program than some lesser features.

A final problem was that none of the evaluation guidelines took into account variations in teaching strategy. Instead the criteria were offered to be applied to all software materials regardless if they used a simulation, drill/practice, tutorial, or game format. Logically, one might reason that each of these strategies would utilize the capabilities of the computer in a slightly different way and therefore certain criteria would be less important or not important at all in an evaluation of that strategy.

In short, there are many problems with these guidelines that require more thinking and certainly more research. Taken separately, each of the problems mentioned above might be examined and alternative procedures recommended:

1. Validity of the Criteria

Probably the most significant problem plaguing the development of evlauation criteria is that, for the most part, few if any of the criteria have been validated in any sort of well conducted research study. We do know from research that some features of an instructional program can affect learner performance, but these are few and are often poorly documented. Most of the criteria included on evaluation guidelines are based on what the developer THINKS contributes to an effective program. This same problem was inherent in most teacher education research until correlational studies began to establish clear relationships between specific teaching behaviors and student achievement. Until these same methods are applied to instruction delivered through a computer-based medium, the problem of validity of criteria will remain a significant one.

One need not look far to find examples to illustrate this point. One criteria which invariably appears on evaluation guidelines is, "Is the program interactive," or "Does the program provide interactions." Few educators would argue that interaction is a valuable characteristic of ANY form of instruction, but it is particularly important when using the full power of the computer-based system. Unfortunately, however, there is wide variability in the definition of interaction. In a program I evaluated recently the designer posed the question, "In what year was Texas admitted to the Union?" As my response, I was supposed to enter a date. So as an experiment I entered "2,000,000 B.C." The feedback informed me, "No, Try Again." I then entered, "4,000,456 A.D." The feedback urged me to "Try Again." Finally, I got serious and entered, "1843." The computer told me, "No. Try Again." For some time I really tried to get the right answer; in fact, at one point I just began entering numbers consecutively, "1846," then, "1847," then, "1848," and so on. Each time my effort was rewarded with, "No. Try Again."

Now, this program is certainly "interactive" and by the definition implied in most of the evaluation guidelines I have seen, it would receive a positive evaluation on that basis. However, the program is clearly terrible in that it provides the learners with absolutely no information to help find the correct answer. Instead, it only serves to increase frustration, inhibit creativity, and stifle the learner.

The validity of other criteria is less subtle. One author offers this criterion, "Does the program offer paging (not scrolling)?" In my opinion, a frame presented in a page format is much more appealing visually than text scrolling up from the bottom of a CRT. But whether that feature contributes more to the effectiveness of a segment of instruction than scrolling does is dubious. But it is this exact type of criteria that is offered on evaluation guidelines as absolute standards of excellence. As individuals interested in developing and encouraging the development of good software, we must look carefully at these criteria and encourage more validation and research. This can only be accomplished by finding positive correlations between various features of programs and learner achievement.

2. Weighting of Criteria

A second problem which seems to reoccur on evaluation guidelines is that in almost all cases criteria are given equal weight in the evaluation process. That is, each criterion is given equal importance in evaluating the software. Obviously, not all features of a piece of software contribute equally to the effectiveness of the total program. With this in mind, it would seem logical that evaluation guidelines be structured so that a composite rating put more emphasis on a feature such as, "interactiveness" than on something like, "Feedback is personalized." Evaluative criteria, therefore, not only need validation, they need to be categorized into a hierarchy which will reflect the degree of contribution each criterion makes to the effectiveness of the overall program.

3. Variation in Teaching Strategy

A final problem seen quite often in evalutive guidelines is that criteria are usually designed to be applied to all software programs regardless of the teaching strategy used to deliver the content. Classroom teaching is very often evaluated without regard for the context of the objectives taught in the lesson. Few educators would argue, however, that very different skills are needed to make individual strategies such as inquiry, lecture, drill, and group discussion effective independently. It would seem to follow, then, that the same criteria applied to evaluating a drill/practice software program might be inappropriate when applying them to evaluate the effectiveness of a computer-based simulation.

Qualifications of Reviewers

In the Blum (1982) study a wide variance in ratings of software were found when a single piece of software was reviewed by three or more raters. Blum attributed this variance to two major factors:

1. Subjective judgments of the evaluators on items that were highly inferential, e.g. "the presentation was boring."

This conclusion only serves to verify the need for objective criteria which are validated empirically through correlational research models. Blum writes,

"There is basically little that can be done to reduce variance in evaluation studies when judgements have to be made as they are based on personal opinion and retraining would not alleviate the problem." (p. 27) To a great extent, personal bias will never be eliminated from any evaluation process. However, if we can find and validate objective criteria which rely on low inference (e.g. "Feedback helped shape behavior in the direction of the desired learning outcome.") rather than high inference (e.g. "The program provides adequate feedback."), much of the bias in evaluation that Blum found might be reduced.

2. Error in scoring due to inadequate training or background of the reviewers themselves.

In the Blum study, an evaluation instrument was developed which systematically analyzed software in terms of its intent, contents, methodology and means of evaluation. Unlike many others, this instrument also evaluated the software in terms of the strategy it used.

Blum found that what often occurred was that one rater would respond to the software from the perspective of his/her background while the other would respond from a different background. A mathematics teacher, for example, would look carefully at content in his/her evaluation whereas a rater with instructional design expertise would assess the software from that perspective. Blum's conclusion was the "There is no way to ensure that training and retraining will acount for varying background factors so that this (variance) can be reduced." (p. 26)

The solution to this problem seems so obvious that I feel that I have perhaps missed the significance of Blum's conclusion. It would seem that one way to achieve greater reliability among raters would be to have raters from similar backgrounds evaluate the courseware. That is, allow three reviewers who have designed software for the English language arts or for mathematics or for whatever each evaluate the same program. This might provide a certain degree of reliability that is currently lacking.

Another solution to this problem would be to choose raters who had actually designed software so that they had that experience in common as well. Individuals who have tried the laborious and frustrating experience of actually creating software often have a deeper appreciation of what an author or designer was attempting in the program. A final solution might be to have each software program rated by three individuals and then report the composite rating rather than let the review fall on the shoulders of just one reviewer.

Lack of Instructional Models

A final problem in evaluating software stems from a general lack of knowledge about how software should be used in a classroom. As students we have had the benefit of hundreds of hours of instruction. Each lesson was in itself a model of how (or how not) to teach a lesson. As teachers we emulated the models we found most useful. In using computer-based education, however, few such models exist. Few of us have ever experienced a computer-based course so have little insight into potential problems. As a result, we often evaluate a software program on the basis of the diskette or computer component itself instead of looking at it as a total system which includes print materials or other media integrated into the system.

These issues, therefore, seem critical to the establishment of valid criteria for evaluating educational software. Tens of thousands of dollars are being invested in software development. It is important that the production of that software is guided by characteristics which have a relationship to learner achievement so that the consum ers of that software can make reasoned choices about the software programs they buy.

REFERENCES

Blum, B.L. "Evaluation of Educational Softward for Microcomputers: An ANalytical Approach," Paper presented at the American Educational Research Association Annuual Meeting, New York, 1982.

MICRO-NETWORKING: SOME PRACTICAL APPLICATIONS

By David R. Rieger

Department of Special Education/Related Services
Johnson County Public Schools, Buffalo and Kaycee, Wyoming

ABSTRACT

This paper deals with the practical aspects of using a networking system for microcomputers. Advantages and disadvantages of networking as well as the author's reasons for selecting a disk-sharing system over the standard floppy disk format for school use are delt with. The Corvus Omninet Networking System is described at length and the positive aspects of hard disk technology are discussed.

INTRODUCTION

For today's educators, the use of a disk sharing system for microcomputers can be both convenient and cost efficient. Some advantages and disadvantages have been identified by Fisher (1982).
Advantages include:
* The use of disk sharing can save the educator time and energy. Since all data is located on one disk, management becomes very simple.
* The teacher has more control. With some systems, one can restrict access to parts of the memory. This allows the educator to manage program or file usage.
* A great deal of money can be saved. Instead of providing memory devices (or disk drives) for each microcomputer station, a central drive is shared. Also, other peripherals can be shared.

Disadvantages include:
* Disk sharing systems are not portable. The microcomputer stations have to be wired into the system.
* Not all systems restrict access to memory. Some students might cause damage to files or completely destroy some data.
* Some disk sharing systems are not easy to use. Additional training to allow teachers to learn how the system works would have to be provided.
* Technical limitations might cause considerable problems. Some systems allow only one student access to a file at a time. Other systems have different limitations.

In this paper, two aspects of disk sharing will be discussed: the first part will deal with how a hard disk network works. In this study, careful attention will be given the Corvus Hard Disk Network System. A complete overview of the system will be presented and explanation will be given on how it differs from a standard floppy disk system connected to just one microcomputer. The second part will deal with some of the application problems that were encountered in setting up such a system. A detailed account will be given to the process of acquiring software for a network system.

The Corvus Hard Disk Network System is only one of several disk sharing systems on the market at this time. For the purposes of this paper only the Corvus Omninet System will be discussed. This results from the author's opinion that the Corvus is a superior product and from the author's belief that the Corvus Hard Disk Network System has the best practical application of disk sharing for public education, Grades K-12. (Note: The Apple II+ was selected as the type of microcomputer for use due to the large amount of software available for it. The Corvus Hard Disk Network System and Apple II+ Microcomputers are completely compatible.)

The author's experience with disk sharing stems from the use of a Corvus Hard Disk Networking System. The system was purchased for a Title IV-C Study to investigate the effect of computer assisted instruction (CAI) on the learning achievement of handicapped students, Grades 1-5. A method was needed that would allow special education teachers to have access to a vast number of computer programs. The standard method of individual floppy disk drives and floppy disks for use with each microcomputer was found to be less than desirable for several reasons:
* The teachers would have to share the programs on floppy disks so that only one program could be used by one microcomputer at a time.
* More teacher control would have to be used to insure that the disks would not be managed by the handicapped students.
* The goal of having handicapped students control a great portion of their own education could not be reached due to the increased teacher control of the software.

The desirable aspects of using a disk sharing system include;
* All programs would be available to every computer on the network.
* Teachers would not have to act as policemen to safeguard the floppy disks.
* Students could access programs by themselves.
* The general availability of a vast library of software on the network would allow the teachers

and students much greater latitude in using the computer.

For the purpose of this paper, the author assumes that the reader has little or no experience with networking systems and has therefore presented the workings of the Corvus Omninet System in detail.

HOW A NETWORK SYSTEM WORKS

The normal configuration of a microcomputer station (Figure 1) includes the microcomputer (which also includes such parts as the central processing unit - CPU, read only memory chips - ROM, random access memory chips - RAM, and support circuitry) a monitor, and memory storage - such as a floppy disk drive. The basic difference between this normal configuration and the use of a network is the memory storage; the network uses a shared memory device. Rather than access memory on separate devices, the network uses a single memory device. In the case of the Corvus Omninet System, a hard disk is used (Figure 2). All of the independent microcomputers are wired together on a common network (Figure 3).

The Corvus Omninet System is controlled by a device called the disk server (Figure 4). Each computer is linked to the disk server by wires. Software on the hard disk allows each computer to call into the network via a user name and password. The disk server acts like a traffic controller in that it allows each user access to only certain parts of the hard disk and to use those parts in either read, write, or read/write modes. If more than one computer calls for access to the hard disk at one time, the disk server will serve the first user while the rest wait. When the shared wires are free, the next user is served. Although this might seem to take a long time, the hard disk works so fast that all users can be served very quickly. The disk server also remembers the location from the "scratch" if another user logs on to the system. Therefore, each system user seems to have private use of the hard disk.

The wiring, mentioned above, is very simple to install and is very inexpensive. Unlike the multi-strand cables that have been used in the past to connect computers, the Omninet System uses a simple twisted pair of wires. All the wiring is configured in a line with each computer tied in at whatever point is chosen. The total distance the system can serve is 4000 feet - 2000 feet on each side of the disk server and hard disk. This distance is most adequate for most K-12 schools.

Each computer has a "tap" off of the main wire with a "tapbox" (Figure 5). The tap box contains small wire clamps that allows for the easy installation of "tap cables" (Figure 6). These are the wires that connect the computers to the main wire.

For use with Apple II+ Microcomputers, a "transporter card" (Figure 7) - shown here (on the top) with a standard Apple Disk II Controller Card - is installed in slot #6 of the Apple (Figure 8). This slot is accessed automatically when the Apple is turned on, so the user is booted into the omninet system by simply turning everything on. (This assumes that the hard disk is already on and ready to receive data.)

The network asks for a user name and password before a user can have access to any part of the hard disk. With different names and passwords, some security is provided. With an application of the network system in an elementary school, a simple user name/password was developed that allowed the user "read access" to most everything on the hard disk. This name/password was "work/go".

With the basic network system by Corvus now explained, an explanation of some of the practical applications, problems, and positive uses will follow.

MICRO-NETWORKING: SOME PROS AND CONS

Negative Items

A variety of problems developed with the installation of the Corvus Hard Disk System. Other problems developed in securing software to place on the hard disk.

One of the problems that took a great deal of time to solve was faulty hardware. Over a period of a month and with much hair pulling and with unnecessary trips to computer dealers and with many telephone calls across the country it was finally found out why several Apple II+ Microcomputers would not work with the Omninet System, yet would work perfectly well with floppy disk drives. The Apple Computer Company has a habit of improving their hardware during the actual production of the product. The changes in the hardware do not effect he compatability of the Apple components. But in this case, newer Apple Microcomputers would not work with Corvus Transporter Cards. The resolution of this problem came with an up-grading of the transporter cards. The problem was very simple to solve, but sales people at local computer suppliers (where the equipment was purchased) provided little if any help.

Another major problem was the time it took to learn how the system worked. As the system manager, or trouble-shooter, I spend many hours to become familiar with the hard/software of the network. The manuals were not very helpful in providing this help and most skills were gained through trial and error. Again, with little or no support, the process of setting up a network can be difficult.

The most difficult problem with the system developed through the manner in which the system was to be used. The Title IV-B project called for the system to use commerically available software. I found out that most software producers are very new to the business and what I was asking them to do had never entered their minds. The Corvus Omninet System must have uncopy-locked software in order for it to be placed on the hard disk. This is due to the fact that a copy must be made of the

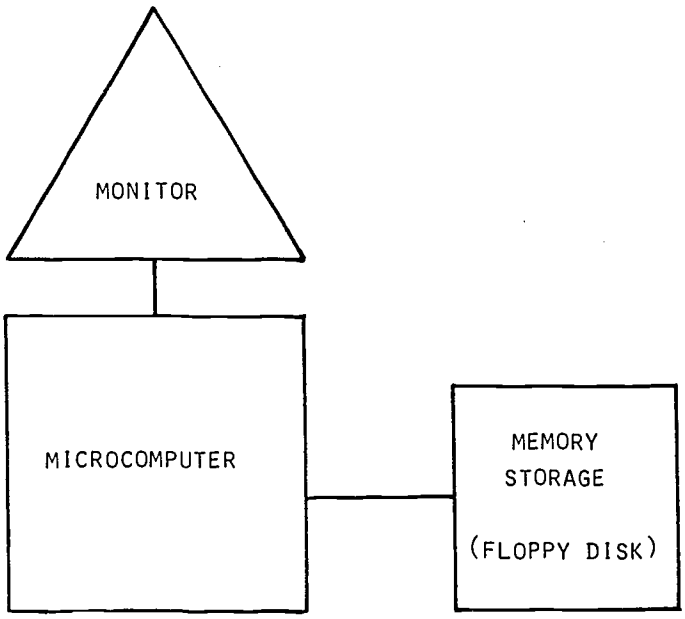


Figure 1. Normal Configuration of a Microcomputer

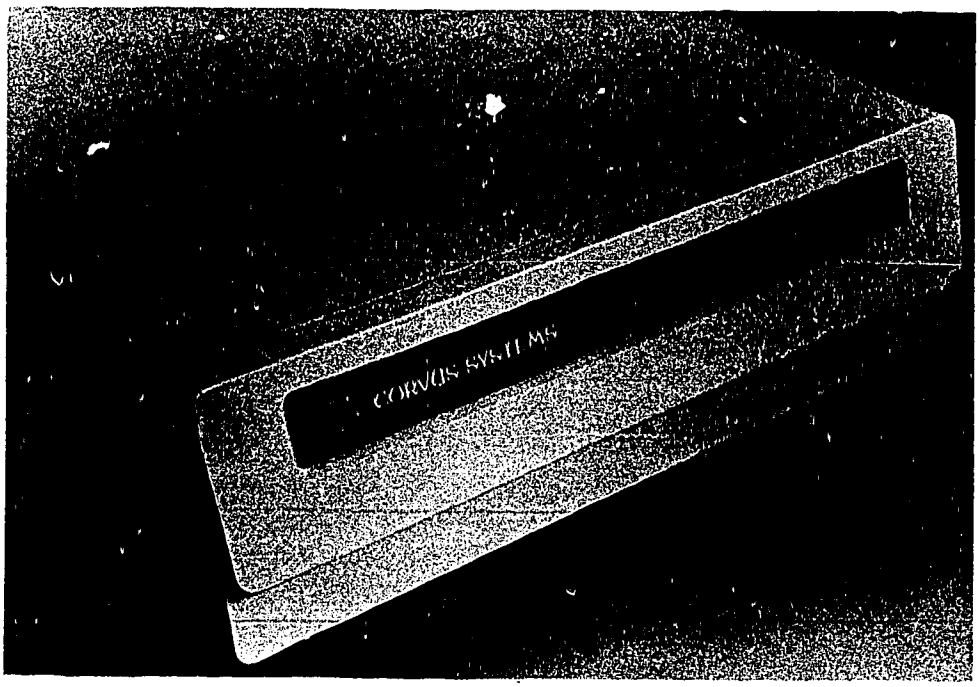


Figure 2. Corvus Hard Disk

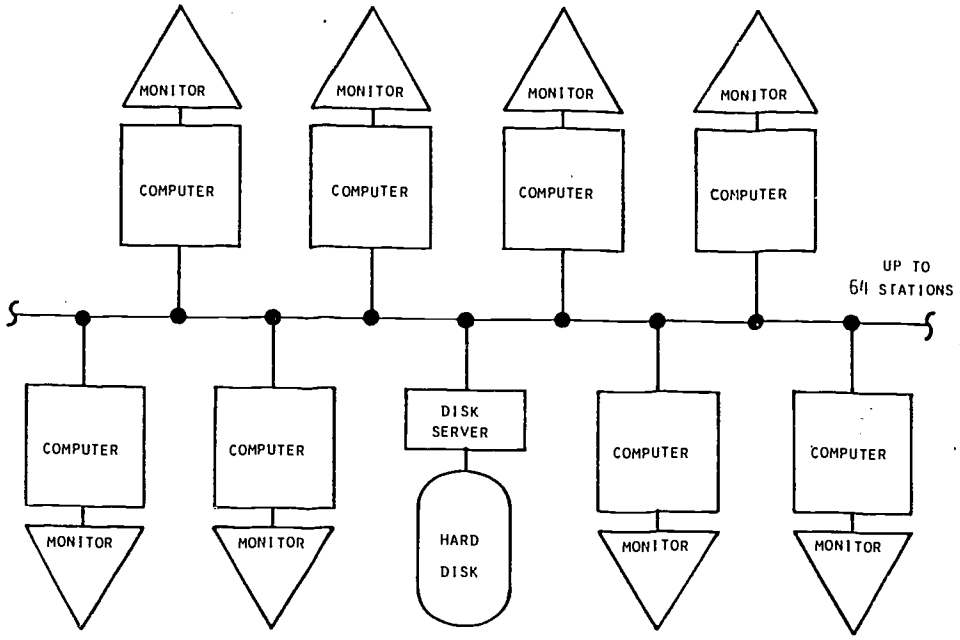


Figure 3. Shared Memory Via Corvus Omninet Hard Disk Network

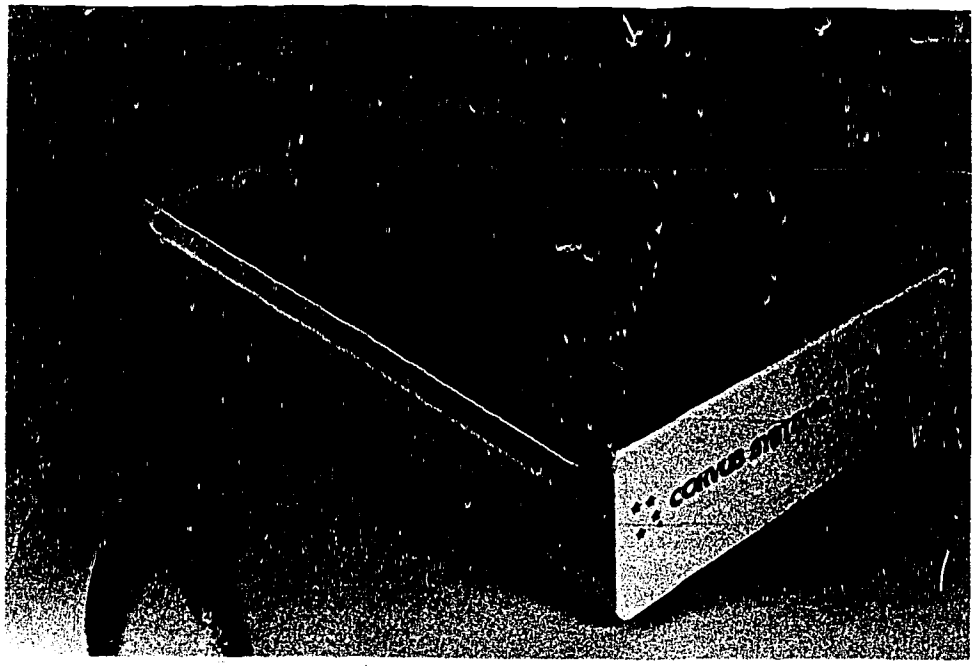


Figure 4. Disk Server

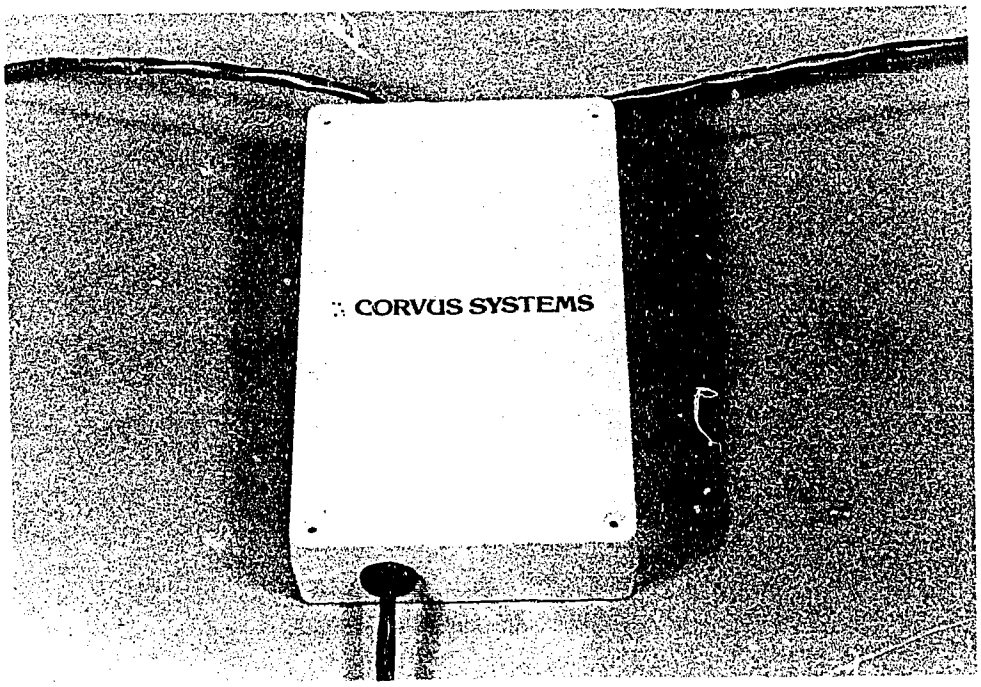


Figure 5. Tap Box

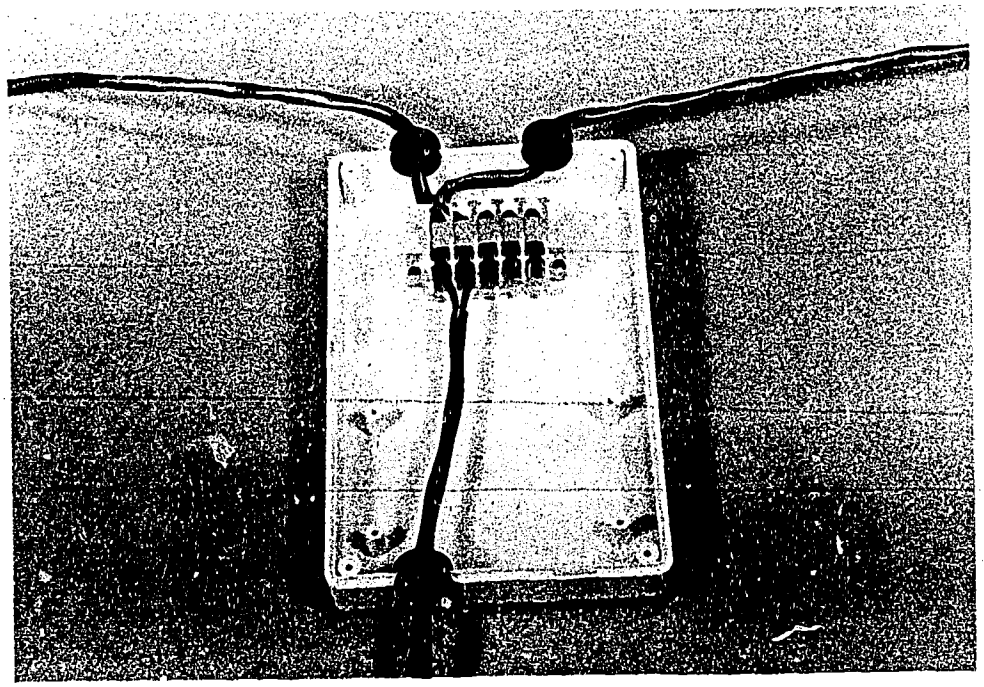


Figure 6. Open Tap Box Showing Connections

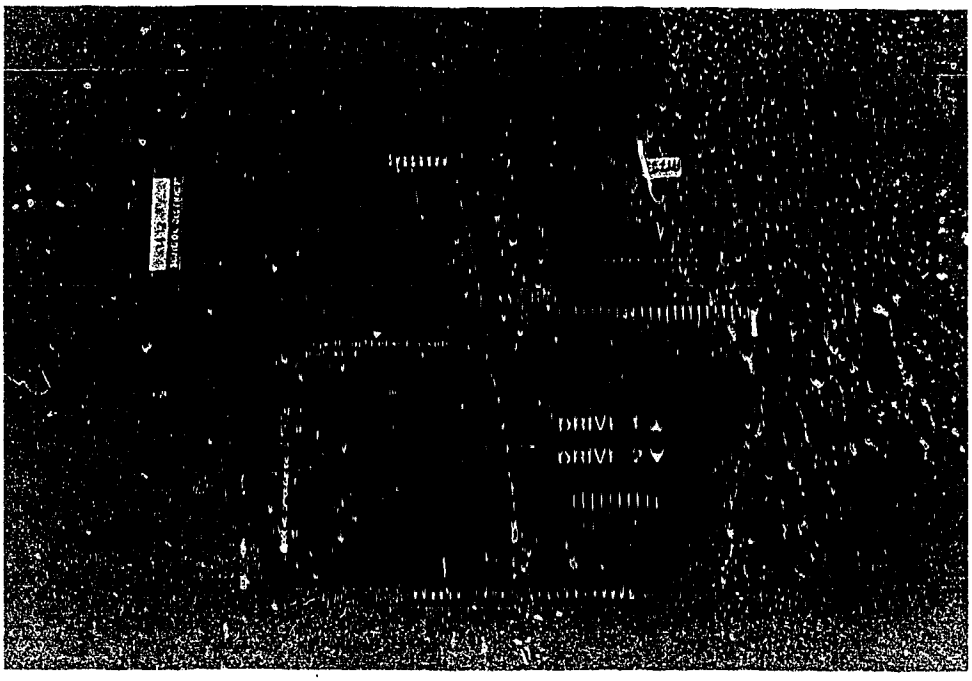


Figure 7. Transporter Card (on top) with Apple II+ Disk Controller Card.

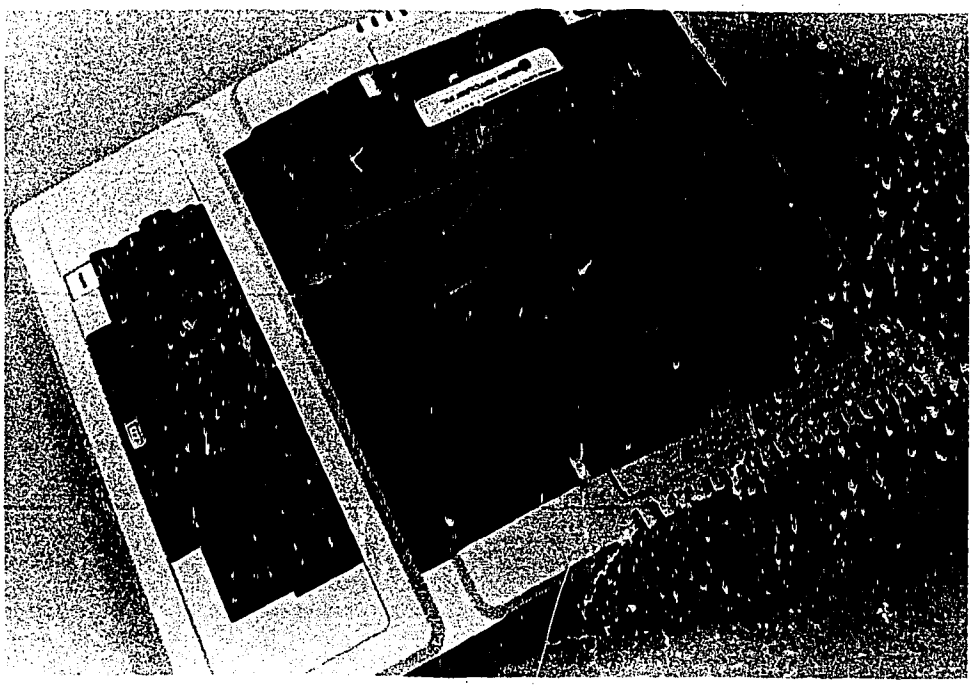


Figure 8. Transporter Card in Slot #6 Inside Apple II+ Microcomputer

program from a floppy disk to a volume on the hard disk. Most producers are very concerned that their software might be stolen by unscrupulous computer users so that to them my need was almost heresy. I prepared a form that stated my needs and my intentions to preserve the copyright of their material. This seemed to help some software producers in allowing me to purchase their materials, but their prices ranged from standard list price to twice the list price. But whatever the price, I was allowed to place a copy of a particular program on ONE hard disk system at ONE location.

Another disadvantage of the networking system is the limitation of space on the hard disk. Although the unit I was using had a capacity of 20 million bytes (approximately 134 - 5 1/4" floppy diskettes worth), the space is not interchangeable. A floppy disk drive allows for unlimited use of different software. The user simply places different floppy disks in the drive. With a hard disk, the actual disk is captive and cannot be exchanged or replaced with another. Data has to be copied to the hard disk. When it is recorded to the hard disk, this data is more or less permanent. It can be removed or replaced, but this again is a permanent change. I found that 20 MB is not all that much space. The Corvus Hard Disk Units allow for up to 4 - 20 MB units to be "daisy chained" together, thereby creating up to 80 MB of storage space on a hard disk network.

I found the difference between a hard disk and a floppy disk system to be much like the difference between a record player and a juke box! A juke box has a limited number of records that are installed in the device. A user can very easily "call up" any record. They are kept neat and clean. But the user has a limited number of "selections" for use at any one time. On the other hand, the record player does not keep the records neat and clean. The user has to handle the records and keep them someplace and in some sort of order. But there are unlimited numbers of "selections" that can be made by the user; one has only to change them.

The hard disk system works just like this juke box. The user is limited in what he or she may access from the system, but the data on the system is extremely easy to get at and use. In order to replace data on the hard disk, quite a bit of work has to be performed.

Positive Items

The advantages of the hard disk network include all of the positive working features of the Winchester-Type Hard Disk, as opposed to a standard hard disk. These features are especially good for several reasons: the Winchester Hard Disk is sealed from the atmosphere and is not affected by dirt, dust, or damage from handling; the Winchester Hard Disk accesses information from and to the disk very quickly (as fast as 500,00 bytes per second can be up or down loaded); the Winchester Hard Disk has few moving parts and normal servicing is rated in years of continuous operation rather than in hours, weeks, or months; and the Winchester Hard Disk does not wear the surface of the disk; the head floats just above the surface on a cushion of air thereby extending the life of the disk indefinitely. Let us look at each of these features in greater depth.

Sealed Disk (Figure 9) - The entire disk/head assembly of the Winchester Hard Disk is sealed in a plastic case that has a super-clean atmosphere. The disk assembly turns inside the case and the heads reach out over and under the disks by a servo arm. This assembly is also enclosed inside the case. With the use of the sealed assembly one of the major problems of floppy disks is eliminated: dirt and grime on the disk.

Fast Access Speed - Since the networking system is based on sharing the memory device, high speed is very important. The fast access time of the Winchester Hard Disk is also a characteristic of all hard disks. The hard disk operates somewhere in the neighborhood of 4 to 6 times the speed of a floppy disk drive.

Few Moving Parts - Since the only moving parts are the disks and the heads, great care is taken to make these extremely reliable. With such few parts to go bad, many Corvus Hard Disk users simply leave the unit on all of the time.

No Disk Wear - Each of the heads that read and write information to the surface of the hard disk does not actually touch the surface as in a floppy disk drive. An air cushion under or over the head floats the head about one micron off the surface: a distance close enough to transfer data but far enough to eliminate wear to the surface of the hard disk.

One of the major reasons that attracted me to the hard disk networking system is still very viable: teachers who want to access a particular program can have almost instant access to it. Since all of the programs are contained on the hard disk and since all the microcomputers on the network have access to the hard disk; each microcomputer can have a catalog of over 130 volumes.

This feature is especially useful if there is a "core of instructional and/or management programs that all users would need to use. Programs for the K-12 setting might include attendance, inventory, student records, etc. for management; while those that support the curriculum by a relationship to a school-wide text book or basic skills program.

One last positive feature of the network system is the ability to expand the system to include many computers (up to 63 total). In Figure 10 we can see what a network configuration for a single school could look like. There is just one memory storage device that is connected to all of the computer stations. In addition, peripherals such as printers could also be shared. One or two

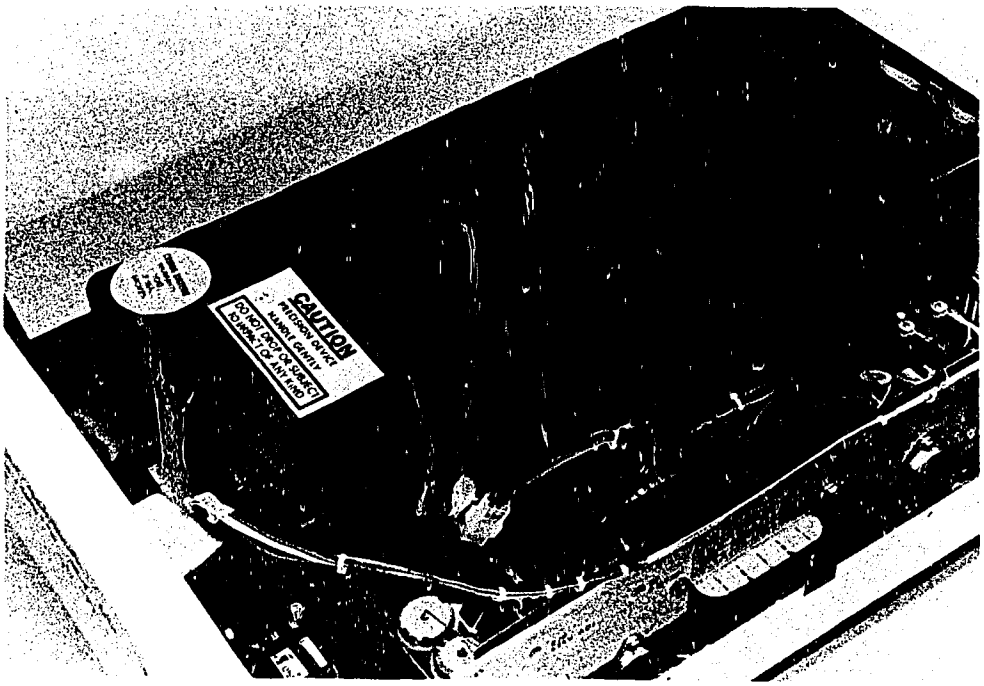


Figure 9. Corvus Hard Disk Drive Showing Sealed Disk Case at Top

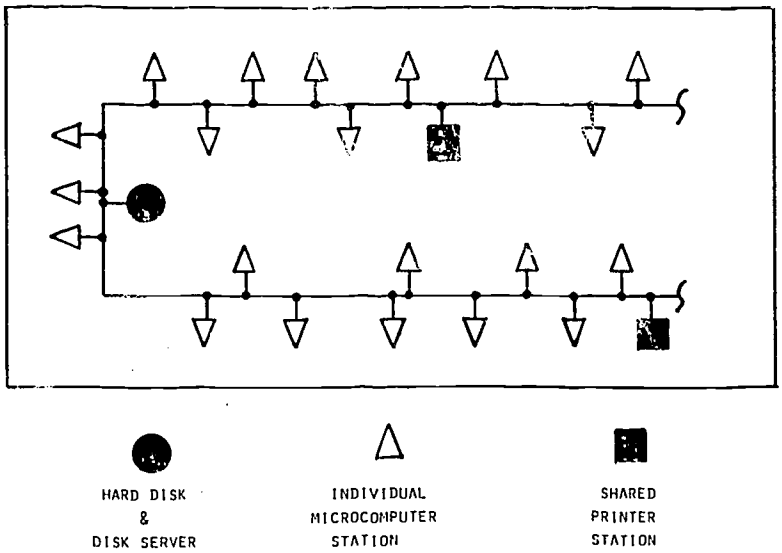


Figure 10. Network Configuration for a Typical School

printers could serve a whole school. The hard disk serves as a buffer and can store data to be printed. If several items are to be printed at the same time, the first item is printed while the rest are ordered and held in the buffer. As each item is printed, the buffer allows successive items to be printed.

CONCLUSION

Although there are many different types of disk sharing systems (Fisher, 1982), the Corvus Omninet Hard Disk System has many positive features that make it very usable in a public school, grades K-12. Micro-networking has some negative features, such as the difficulity of obtaining uncopy-locked software; but these are more than amply offset by such positive features as providing a large library of software to all microcomputer stations at all times, expandability at low cost, and extremely good reliability of the hardware.

REFERENCE

Fisher, Glenn. Disk Sharing: How To Make One Disk Go "Round. "Electronic Learning", 1982, 1, 46-51.

LIST OF FIGURES

Computers in the Elementary and Secondary Mathematics Education

Sheldon P. Gordon
Suffolk County Community College

ABSTRACT

This session will focus on the uses of computers in elementary and secondary mathematics education with emphasis on developing and implementing such usage.

The University of Delaware has developed extensive programs to work with elementary and secondary schools in enhancing the use of computers throughout the curriculum with special focus on mathematics. As part of the present session, a report on one phases of these efforts involving a project to provide computer literacy for preservice elementary education majors will be described. In this project, each student is given the opportunity to become familiar with the PLATO computer system and a variety of microcomputers. The presentation will consist of program description and sample lessons developed for the Apple II computer.

In New York City, the high schools have developed a variety of curricula in computer literacy and computer mathematics. The variety of curricula will be reviewed by looking intensively at the structures designed and successfully implemented in several different schools.

In one school, the entire 9-11 Mathematics curriculum has been restructured to allow infusion of computer problem solving and computer programming in BASIC throughout the curriculum. Students are introduced to various programming techniques as needed to solve the problems encountered in the traditional curriculum. The philosophy behind this approach is that the school should teach mathematics and problem solving, not computer programming.

A second school, operating under the same basic philosophy, has approached curriculum development in another way. The faculty chose to keep programming entirely separate from the traditional mathematics curriculum. Their programming courses, however, are still mathematics courses. Instruction is in the mathematical concepts with computer solutions growing out of the mathematical solutions.

PARTICIPANTS:

William B. Moody
University of Delaware

Neal Ehrenberg
New York City Board of Education
Project Director for Computers, Technology and Research

AUTHOR INDEX