INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University Microfilms International
300 N. Zeeb Road
Ann Arbor, MI 48106

Arch, John Clinton

A MODEL FOR DEVELOPING AN ELEMENTARY SCHOOL COMPUTER SCIENCE CURRICULUM

University of Oregon PH.D. 1984

A MODEL FOR DEVELOPING AN ELEMENTARY SCHOOL COMPUTER SCIENCE CURRICULUM

by

JOHN CLINTON ARCH

Presented to the Division of Teacher Education
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

March 1984

APPROVED: David Moursand

Mr. David Moursand

An Abstract of the Dissertation of

John Clinton Arch for the degree of Doctor of Philosophy in the Division of Teacher Education to be taken March 1984.

Title: A MODEL FOR DEVELOPING AN ELEMENTARY SCHOOL COMPUTER SCIENCE CURRICULUM

Approved: David Moursund
David Moursund

The purpose of this dissertation is to present a model for developing an elementary school computer science curriculum which concentrates upon teaching the problem-solving skills commonly used by computer scientists.

The model is presented in the following six steps:

(1) DEFINE THE PROBLEM-SOLVING METHODS COMMONLY USED BY COMPUTER SCIENTISTS. This was accomplished by interviewing ten computer science professors, six at the University of Oregon and four at the University of Houston.

(2) DEVELOP ACTIVITIES TO TEACH THESE PROBLEM-SOLVING SKILLS TO ELEMENTARY SCHOOL STUDENTS. Activities were obtained from several sources, including journals and magazines, teachers observed in their classrooms, fellow graduate students, and books. When an activity could

not be found, the author developed a suitable activity.

(3) FIELD TEST THE ACTIVITIES. The test sites were two elementary schools in Eugene, Oregon. Fourth and fifth grade students were used.

(4) INTERVIEW STUDENTS. The students were interviewed to discover whether or not they liked the activities.

(5) INTERVIEW TEACHERS. The activities were shown to 20 teachers, 12 of whom had gone through an intensive 296-hour training program in computer education, and eight regular classroom teachers without special computer training.

(6) DEVELOP A WORKSHOP. A one-day workshop was developed to teach teachers how to use the activities. The workshop was taught to a group of teachers in the Houston Independent School District.

In addition to outlining the model, this dissertation presents the data collected from a pilot study of the implementation which was carried out in Eugene, Oregon, and Houston, Texas.

VITA

NAME OF AUTHOR: John Clinton Arch

PLACE OF BIRTH: Freeport, New York, U.S.A.

DATE OF BIRTH: August 30, 1946

POST-SECONDARY SCHOOLS ATTENDED AND DEGREES AWARDED:

Columbia College (A.B. 1968)
University of Arizona (M.Ed. 1971)
Vanderbilt University (J.D. 1975)

AREA OF SPECIAL INTEREST:

Elementary School Computer Science Curriculum

PROFESSIONAL EXPERIENCE:

Teacher:

Greek American Institute, Bronx, New York (1968-69), grades 5-8 Science

P.S. 208, New York City (1969-70), sixth grade

Metropolitan Board of Education, Nashville, Tennessee (1973-80), grades 5 and 6

Student Attorney, Legal Aid, Nashville, Tennessee (1972)

Graduate Teaching Fellow, University of Oregon (1982-83)

Author (1975-Present)

Tax Consultant (1975-Present)

Director, Computer Literacy Project, Department of Technology, Houston Independent School District, Houston, Texas (1983-Present)

AWARDS AND HONORS

Awarded a grant of $160,000 by the United States Department of Education to develop a secondary school computer literacy curriculum. The grant proposal was written by Susan Sclafani, Richard Smith, and me in August of 1983.

PUBLICATIONS

Arch, John C., NEA Federal Income Tax Guide for Educators, National Education Association, published 1975 - 1983.

Arch, John C., book review of The Complete Computer, published in the September issue of The Computing Teacher, ICCE, University of Oregon, Eugene, Oregon, 1983.

ACKNOWLEDGEMENTS

To my wife, Diane, and our daughters, Joanna and Allison, and to the memory of my Father, John F. Arch.

TABLE OF CONTENTS

APPENDICES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

Purposes of Study

The purposes of this dissertation are to:

(1) propose a model for developing a computer science curriculum for elementary school students that exposes them to the essential problem-solving skills of computer science;

(2) develop and pilot test a two to three-week elementary school computer science unit by following the model proposed in (1);

(3) develop and pilot test a workshop to train elementary school teachers to teach the unit developed in (2);

(4) accomplish (1) through (3) using a relatively minimum amount of hardware.

Defining the Problem

There can be little doubt that more funding for computer science education is coming to our schools.

Recently, the Governor of Tennessee, Lamar Alexander, called for increased expenditures on computer science education in that state, with the eventual goal of placing at least one microcomputer in every school in the state (The Tennessean, March 13, 1983). This in a state not known for leadership in the area of technological education.

The federal government is also making increased financial commitments to upgrading technological education. For example, the Emergency Mathematics and Science Education bill, recently passed by the House, would give $250 million to schools to improve science and math curricula (New York Times, March 27, 1983).

In 1982, the Houston Independent School District created the nation's first Department of Technology. With an annual budget of over $4 million and a staff of 40, this Department has the dual tasks of overseeing the introduction of computer science education to Houston's 190,000 students and training to the District's 14,000 teachers (Sturdivant, 1983). The Houston Independent School District allocated $3,413,856 for the Department of Technology, an average expenditure of $17.60 for each of the District's approximately 194,000 students (Department of Technology, 1982).

A problem for computer science educators is to determine the best method for spending their share of this increased funding.

To date, the thrust of instructional computing has been threefold, as perhaps best formalized by Robert Taylor's (1980) book, Tool, Tutor, Tutee. For the most part, students who have been exposed to computers have been exposed in one or more of the following ways: (1) they have been taught to program (the computer as tutee); (2) they have been taught to use the computer as a powerful problem-solving tool (the computer as tool); or, (3) they have been taught by the computer through some form of computer-assisted instruction (the computer as tutor).

The common thread running through these three strands is the computer itself. Each requires students to sit in front of a terminal or microcomputer. Aside from the fact that it will be extremely expensive to buy the hardware and software needed to fully develop these uses of the computer, each of these approaches misses what may be the key element in the development of a computer literate populace, namely the ability to think like a computer scientist. In their rush to use the computer for instructional purposes, many people have overlooked this important aspect of computer science.

Problem Solving and Computer Science

Problem solving is currently a "hot" topic in education.

However, as the literature search in Chapter 2 reveals, there is no consensus either on what is meant by the term "problem solving" or how to teach general problem-solving skills. This dissertation proceeds on the assumption that all academic disciplines, computer science among them, require the mastering of a specific set of problem-solving skills. No attempt is made to "prove" the existence of a general problem or skill or series of skills that can be learned by studying computer science and then transferred to other areas. Such skills may in fact exist, but it is not a goal of this dissertation to find them.

One possible reason computer science has developed so rapidly into a separate discipline is that it requires its own unique set of problem-solving skills. Indeed, it has been claimed that one purpose of professional education is to train problem solvers in particular fields (Cyert, 1980). Professor Moursund has written:

> Any discipline can be framed as a hierarchic set of problems to be solved. Instruction in a discipline leads to understanding the nature of its problems--problems that have been solved, and ways to solve some of the problems; what problems have not been solved, and ways to formulate and attack new problems (Moursand, 1981, Intro, p. 88).

Computer science cannot adequately be taught simply as an adjunct to some other discipline, such as mathematics. It must be recognized that computer scientists solve problems in different ways than do mathematicians, writers, lawyers,

chemists, and so on. It is a belief expressed repeatedly throughout this dissertation that these problem-solving skills are a key element of what is essential about computer science. Furthermore, it is contended here that these problem-solving skills can be defined, isolated, and distilled into a curriculum that elementary school students can understand and that this curriculum can be taught without the purchase of a great deal of expensive hardware and software. This dissertation will outline a procedure for developing such a curriculum.

The Ideal

The ideal situation for delivering computer science education to elementary students would be to have enough computers and trained personnel to provide every student with as much access to computers as needed. The District should maintain a large software library and a staff of experts to provide sufficient inservice training to keep the teachers current in their knowledge of computers. The problem with reaching this ideal is twofold--money and time; a great deal of money would be required to buy the requisite hardware, and a great deal of time and money would be needed to train the necessary personnel. Eventually, this ideal will be attained but not in the immediate future. It is clear that an interim solution is needed, one that can expose young children to the

problem-solving skills required by computer scientists without requiring schools to purchase too much hardware too soon.

Significance of this Topic

If the model proposed here is developed successfully, it would be a step toward making our youngest generation more technologically sophisticated. The model would provide a firm foundation for understanding the problem-solving skills that are required for the efficient use of computers.

More specifically, a unit developed under the model proposed in this dissertation would have the following advantages:

(1) It could be squeezed into the existing curriculum with a minimum of disruption, it being much easier to make room for a ten or fifteen-hour unit than for a term-long course.

(2) Teachers could be trained to use the unit in a fairly short time. (AND, the training would NOT prepare them for higher paying jobs in the private sector, as a unit involving extensive programming and use of computer hardware might).

(3) No large investment in hardware would be required. In fact, many of the activities developed during the pilot test required no computer hardware at all.

(4) Students would be given a brief, yet solid

introduction to a powerful type of experience--the knowledge of thought processes. This would help prepare them not only to think like computer scientists (when so doing is appropriate, of course) but also make them less vulnerable to the inevitable technological changes of the future. The unit would help produce thinkers and problem solvers instead of students whose only knowledge of computers is so hardware-dependent that it would be outdated several years after it has been acquired. By stressing thinking patterns and problem-solving skills rather than hardware manipulation, a curriculum can be developed that is as machine independent as possible. This machine independence will make it much easier to implement than a curriculum that depends upon a particular brand of hardware.

(5) A framework would be provided for maintaining an up-to-date elementary school computer science curriculum. By continuously repeat-the steps of the model, a school district could keep its computer science curriculum up to date.

Summary

In this dissertation, I will propose a model for developing a unit to teach a number of the essential problem-solving skills of computer science to elementary school children. To ensure validity, faculty in a cooperating computer science department should be interviewed. Activities based upon the information gathered in the interviews and appropriate for elementary school children can then be developed, using as resources the experiences and training of local teachers, the ideas of graduate students in computer science education, and the computer science education articles in various magazines and journals. The activities should be field tested and then taught to teachers in inservice workshops. Finally, these teachers should be observed and interviewed periodically as they attempt to implement what they have learned in the workshop.

The goal here will be to present a model for developing a unit to teach the essential skills of computer science to elementary school teachers without requiring schools to purchase an inordinate amount of equipment.

Is there a need for such curriculum? The discipline of computer science is not as well developed at the elementary school level as it is at other levels in our educational system. The Association of Computing Machinery has developed a detailed curriculum for university computer science

departments to help them develop competent computer scientists (ACM, 1978). There have been several curricula developed to guide the implementation of secondary school computer science. There have been no comparable efforts made to develop an elementary school computer science curriculum of the type proposed here. It seems clear that before long something will be done to fill this vacuum. This dissertation is an attempt to present a methodology for doing the job correctly. The curriculum development model proposed and pilot tested in this dissertation is an attempt to provide some structure to the development of an elementary school computer science curriculum. The model will enable elementary schools to incorporate some of the essential problem-solving skills of computer science into their curricula.

After the model has been presented, it will be used to develop an actual unit in computer science for elementary school students. When finished with the unit, students should have better insight into what it takes to think like a computer scientist. Computer science requires specific problem-solving skills that can be taught independently of actually using a computer, which is not the same as saying that the learning of these skills cannot be enhanced by providing students with access to computers. As mentioned previously, the ideal situation would be to have enough hardware for each student to have access to it whenever the

need arose. But, it will not be possible in the immediate future to train enough teachers and purchase enough equipment to do the optimal job. Even if it were, however, the content of such a curriculum would require some guidance. The key ideas of computer science would still have to be defined, activities developed and tested, and personnel trained. This dissertation will outline a method for beginning to attain these goals.

CHAPTER TWO

REVIEW OF LITERATURE

Curriculum Development in General

Much of today's writing on curriculum development can be traced back to Ralph Tyler's book, Basic Principles of Curriculum and Instruction, first published in 1949. In it, Tyler enumerated his now famous rationale for curriculum development:

> Four fundamental questions...must be answered in developing any curriculum and plan of instruction. These are: (1) What educational purposes should the school seek to attain? (2) What educational experiences can be provided that are likely to attain these purposes? (3) How can these educational experiences be effectively organized? (4) How can we determine whether these purposes are being attained? (Tyler, 1949)

Curriculum planning begins with needs assessment and passes through a series of steps before teachers begin implementation. A widely accepted model for developing curriculum was proposed by Taba in 1962. She proposed the following seven steps, based closely on the work of Tyler:

Step 1: Diagnosis of needs

Step 2: Formulation of objectives

Step 3: Selection of content

Step 4: Organization of content

Step 5: Selection of learning experiences

Step 6: Organization of learning experiences

Step 7: Determination of what to evaluate and of the ways and means of doing it.

It has been argued that these and similar models of curriculum development are based too much upon the authors' own opinions rather than upon the findings of rigorous empirical research (Goodlad, 1980). Indeed, Tyler's book contains not one citation, nor even a bibliography. But, as Goodland points out, there has not been a great deal of research into curriculum development. Hence, the plethora of opinion in much of the writing on the subject.

An extreme view is that of Papert (1980), who argues that most of today's curriculum planning is misguided. Papert, expanding upon the theories of Piaget, calls for what he terms a theory of learning without curriculum. To Papert, this new learning means "supporting children as they build their own intellectual structures with materials drawn from the surrounding culture (Papert, p. 32)." He sees the microcomputer in general and LOGO in particular as having the potential to provide a great deal of this needed support.

Despite the shortage of concrete results from the research into curriculum planning and development, it is obvious that many millions of children will arrive at the school house door each school day morning for as far into

the foreseeable future as one cares to look. There a curriculum of some sort will await them. The process we call education will not suspend itself while educators develop a universally accepted definition and model of curriculum (Goodlad, 1980).

Curriculum experts have long debated the role that should be played by scholars in the development of curriculum. At the higher levels of American education, scholars have exerted almost complete control over curriculum. For instance, academic specialists have totally controlled most university curricula. Indirectly, they control much of secondary curriculum, since it is universities that set the standards for the advanced placement and achievement tests (Pratt, 1980).

King and Browell (1966) took the extreme view that only scholars have the training and knowledge to determine what subjects should be taught--not just in universities but in schools at all levels. Others take an opposing tack:

> Scholars, as such, are incompetent to translate scholarly material into curriculum (Schwab, 1973).

To say the least, the state-of-the-art regarding the input of scholars into curriculum development is unsettled. No one, however, has gone so far as to say that scholars should have no input. The consensus seems to be that some input is needed from scholars, but that other legitimate sources of input exist and should be sought out. These other sources of input include:

(1) Skilled and Experienced Teachers, Curriculum Design Experts (Pratt, 1980).

Teachers are seen as essential because of their intimate knowledge of students, the school environment, and what Pratt calls their "practical pedagogy." Teachers may also be used to demonstrate the new curriculum to other teachers, after the design phase has been completed.

A curriculum design expert is needed to keep the other team members on task and to keep the group focused on the forest instead of the trees. Pratt claims that many failed curriculum development projects can be traced to the lack of such a curriculum expert on the design team (Pratt, p. 121).

(2) Administrators, Principals, Classroom Teachers, Supervisors, Lay Citizens (Shuster and Plaghofl, 1977).

Shuster and Plaghofl maintain that administrators are needed for two purposes: (1) to define the limits within which curriculum development must be contained, and (2) to encourage curriculum designers to innovate and experiment within these limits.

Principals are seen as the liaison between the administration and the classroom teacher. Regarding the importance of the teacher, Shuster and Plaghofl contend that:

> The classroom teacher is the most important person in the curriculum improvement program (Shuster and Plaghofl, p. 473).

Teachers need an open mind toward curriculum and must be able

to articulate their curricular needs to their principals and administrators.

Supervisors, by which Shuster and Plaghofl mean district-wide curriculum directors, are responsible for encouraging principals and teachers to constantly evaluate and improve their curricula.

Lay citizens are seen as having a responsibility in a democracy to take an active interest in the education of their children. One of these interests should be overseeing the curriculum their children are experiencing in school.

(3) Teachers, Curriculum Leaders, Students, Parents, Consultants (Wiles and Bondi, 1973).

Wiles and Bondi view teachers as a school system's lightning rods. For instance, they will be the first to know whether or not a particular curriculum is not working as intended. They have firsthand experience about the areas of the existing curriculum that need improving.

Curriculum leaders are persons who have:

> Primary responsibility for planning, coordinating, and/or managing curriculum activities in a school district (Wiles and Bondi, p. 217).

(4) Teachers, Students, Parents, Principals, Central Office Personnel, Outside Resource Experts (Morley, 1973).

Morley holds the opinion that the classroom teacher is essential to curriculum development because:

> It is the classroom teacher who ultimately will and does determine the character of life and learning in the classroom (Morley, p. 17).

Student input should be sought as part of the curriculum development process because students, as the ultimate consumers of the curriculum, can provide insight into their needs and interests.

Parents' input is seen as critical to the success of any new curriculum, especially if the issue is at all controversial.

(5) Learning experts, The Milieus in Which Learning Occurs, Someone Who Understands the Teachers Who are to be Involved, Curriculum Experts (Schwab, 1973).

By learning experts, Schwab means people who are familiar with how children learn. An expert in this field should be able to answer questions such as: What should children find difficult? What are the norms for children in the target age group?

Milieus refers to the social setting in which the learning will take place, including the school, the individual classrooms, and the community.

Knowledge about the teachers who will implement the curriculum should include what they know now, how much training they will need, and so on.

The curriculum expert is seen as overseeing the entire development process by helping to smooth out the difficulties that will surely arise among the other members of the group.

Curriculum Development in Computer Science in Elementary Schools

Many authorities recognize that something must be done to bring computer science education into the elementary schools, but attempts to do so invariably have different goals than what is proposed here.

An important project in the area of elementary school computer science education is AN APPROACH TO INTEGRATING COMPUTER LITERACY INTO THE K-8 GRADES (Hunter, 1980). This project is being funded by the National Science Foundation. Begun in 1980, with a termination date of September, 1983, this project seeks to find, develop, and disseminate information about computers into the nation's elementary schools. The project seeks to:

> ...infuse computer-related skills into the traditional curricula of elementary and junior high school science, social studies, and mathematics courses (Hunter, p. 3).

While the goals of this project are important, they ignore the development of skills that are essential to

developing an understanding of the thought processes and problem-solving skills essential to computer science.

A project developed in 1974 (Robinson, 1974) for the Oklahoma State Department of Vocational and Technical Education had, among other things, the following objectives for students:

(1) to identify the basic parts of a computer;

(2) to apply rules for raising base ten numbers to a power;

(3) to change base ten numbers into base two numbers;

(4) to read a computer card; and,

(5) to write a flow chart.

These may or may not be worth goals, but they certainly do not impart any insight into the thought processes that comprise the foundation of computer science.

A project developed at Stanford University in 1975 came closer to some important issues of computer science (Weyer, 1974). The goals of this project were to teach programming to students between ten and fifteen years old and, by so doing, to impart certain key concepts about computer science. Among the concepts the developers sought to impart were the following:

(1) literals;

(2) names and values;

(3) evaluation and substitution;

(4) stored programs;

(5) decisions;

(6) procedures;

(7) argument passing;

(8) functions;

(9) recursion; and,

(10) iteration.

There are at least two problems with this approach: (1) it requires a staff that is proficient in programming, and (2) it requires too much hardware and software to be practical for all but the wealthiest school districts. Given the shortage of computer science educators and the current state of the economy, these are serious shortcomings.

Another approach might be termed the computer literacy approach. Many curriculum guides mention that students should be taught computer literacy. For instance, the PRISM (Priorities in School Mathematics) Project of the National Council for the Teaching of Mathematics includes computer literacy as one of the nine strands it recommends school stress during the 1980's (PRISM, 1981).

While computer literacy is defined in many ways, it generally includes a brief introduction to computer facts (terminology, history, and so on), an overview of the impact of computers upon society, and a brief introduction to programming (Graham, 1983; Stern and Stern, 1983; Frates and Moldrup, 1983). These are each worthy goals, but is my

belief that knowledge of the thought processes essential to computer science is also a worthy goal and a more attainable goal for the immediate future.

Computer Literacy Projects

Many school districts have implemented or are in the process of implementing computer literacy curricula. The Cupertino Union School District in Cupertino, California, developed and implemented a district-wide computer curriculum in 1981. This curriculum was revised in the fall of 1982 (The Computing Teacher, March, 1983). Implicit in this curriculum is the belief that if students use and study about computers, they will be better able to function in the computer age. The curriculum is divided into the following sections:

(1) Computer Awareness
(2) Computer Interaction Skills
(3) Computer Programming
(4) Social Sciences
(5) Language Arts
(6) Science
(7) Mathematics

This curriculum does not specifically mention the thought patterns required by computer science. The content covers basically a series of how-to competencies. For example:

901. Explain the concept of programming.

902. Apply problem-solving strategies effectively using a computer program.

903. Perform simple procedures using DOS commands to:

903.1 Copy a simple program.

903.2 Save a simple program.

903.3 Delete all or part of a program.

904. Explain simple error messages.

905. Use editing procedures to correct programs.

(The Computing Teacher, March, 1983, p. 8)

In 1982, the Albany School District in Albany, California, developed an elementary school computer awareness curriculum consisting of the following five broad goals:

(1) Use

(2) Programming

(3) Instructional Uses

(4) Parts and Functions

(5) Vocational Uses and Impact

(Fisher, 1983)

These goals may be worthy, but they seem to be narrowly focused. They fail to directly address the issue of the

the thought patterns and problem-solving skills that the computer age will require.

A comprehensive computer literacy curriculum has been developed by Dr. Gary Bitter of Arizona State University. This curriculum divides computer literacy into two broad areas, computer awareness and computer programming, with the former requiring little or no access to computers and the latter requiring quite a bit of access (Bitter, 1982-83).

While this curriculum covers several potentially important topics, such as logic, problem solving with LOGO, and word processing, it gets bogged down in material inappropriate for young children: computer generations, history of computing, types of computers, computer languages, and so on. These topics do not seem important enough to teach to all elementary school students.

These computer literacy curricula seem to be based upon a belief that exposure to computer hardware and programming is sufficient to instill in students the profound changes in thought patterns that the computer age will demand. It is contended in this dissertation that these thought patterns are too important to be taught indirectly. They should be defined before any elementary school computer science curriculum is developed and should become an integral part of any such curriculum.

Professor Dave Moursund claims that over time, the definition of computer literacy has changed. Originally,

computer literacy referred to "an awareness level of computer knowledge (Moursund, Teacher's Guide: 1980, p. 29)." To gain this knowledge, students would read about computers--their components, uses, limitations, and so forth. But, microcomputers have forced a change in the definition.

> Nowadays the goal of universal computer literacy focuses on a working knowledge--a functional level of knowledge--about using computers (Moursund, Teachers Guide: 1980, p. 30).

A key phrase here is "knowledge about using computers." The curricula described above do not focus sharply enough on this component of computer literacy. Too much time is devoted to secondary topics.

Moursund takes this idea a step further in Introduction to Computers in Education for Elementary and Middle School Teachers.

> The computer literate student understands and uses computers as an aid to problem solving (Moursund, Introduction, 1981; p. 89).

Problem solving and computer literacy is an important linkage for the present and future of education. This dissertation does not plan to explore the relationship between problem solving and computer literacy across the entire spectrum of pre-college education. Here, the goal is more modest: to explore the problem-solving skills used by computer scientists and develop a model for defining and developing activities to reinforce these skills.

In Universities

One of the largest computer science curriculum projects undertaken in this country was the Association for Computing Machinery's CURRICULUM '78: RECOMMENDATIONS FOR THE UNDERGRADUATE PROGRAM IN COMPUTER SCIENCE (Communications of the ACM, March, 1979). The final report of the ACM's Curriculum Committee contains a detailed outline of courses for a comprehensive undergraduate curriculum in computer science.

The proposed curriculum, which has been implemented by the University of Oregon's Department of Computer and Information Science, includes eight core courses as well as recommendations for electives, service courses, and so forth. The core courses are designed so that computer science majors who take them should:

(1) be able to write programs in a reasonable amount of time that work correctly, are well documented, and are readable;

(2) be able to determine whether or not they have written a reasonably efficient and well-organized program;

(3) know what general types of problems are amenable to computer solution, and the various tools necessary for solving such problems;

(4) be able to assess the implications of work performed either as an individual or as a member of a team;

(5) understand basic computer architectures; and,

(6) be prepared to pursue in-depth training in one or more application areas or further education in computer science.

(Communications of the ACM, March, 1979, p. 149)

The Curriculum Committee's final report had a total of 83 contributors, of whome 71 were university professors, including all five of the editors. These figures support the contention made previously in this dissertation that at the university level, scholars control the curriculum.

In Secondary Schools

At the secondary level, there have been several recent developments of note. The Elementary and Secondary School Subcommittee of the ACM has published Computer Science in the Secondary Schools: Recommendations for a One-Year Course (Rogers and Austing, 1981). The Committee presents the following goals for a secondary school computer science course:

(1) To provide the student with practice in making appropriate use of computers as tools for problem solving in a variety of circumstances, applying this both to individual and group problems.

(2) To provide the student with a realistic concept of the power, usefulness, and limitations of computers.

(3) To provide the student with knowledge about the role of computers in current information processing and the

effect on social structures of the application of computers.

(4) To provide the student with a context from which to consider possible future directions in computing.

(p. 652)

It is intended by the Committee that this course be made available to all high school students regardless of whether they attend an academic or a vocational school. The course is to have a problem-solving emphasis from the very beginning, with students being taught "problem solving in the most general sense (p.652)." Other topics to be covered include: programming, applications, history of computing, and social and ethical concerns. Prerequisites will include elementary algebra or geometry. While the content of these courses is not a prerequisite for learning computer science, the authors feel that the problem-solving experience that comes from studying math at these levels will be useful for studying problem solving in computer science. However, provisions are included for the teacher to waive even these minimal requirements for certain students.

The International Council for Computers in Education has also published a curriculum guide for a secondary school course in computer science. This booklet, entitled An Introduction to Computers and Computing, is organized into 32 lesson outlines, approximately one per week for an entire academic year. The lessons are composed of four strands:

Applications of Computers, Programming, Computer Environment, and Social Impact. The course has broadranged computer literacy objectives, with the stated goal being: "to provide the student with a broad understanding of what the use of computers and automatic computing means in the world in which the student lives (ICCE, pp. 5, 6)."

Problem Solving

In the area of problem solving, there has been much done but there is still a great deal of uncertainty and confusion. Greeno (Tuma, 1980) believes that students need a knowledge base before they can solve a problem in a particular subject area, but he criticizes most attempts at teaching problem solving as being too narrowly focused on subject matter. I see this narrow focus as a fatal flaw in much of the computer curriculum that has been developed to date. Greeno calls for more effort to be expended in teaching general problem-solving skills which transcend specific subject matter. The problem with this approach is deciding which techniques can be transferred from one type of problem to another. This dissertation presents a methodology for developing a curriculum that will teach some of the main problem-solving techniques of computer science to elementary school children.

Can general problem-solving skills be taught? The answer is that we do not know. Rubinstein has developed a course at UCLA which he claims can train college students in

general problem-solving skills (Rubinstein, 1980), but Reif critizes Rubinstein's lack of empirical evidence to support his claims, despite the fact that the course has been taught for nearly a decade (Reif, 1980). Larkin (1980) claims that research into general problem solving is hampered by the fact that problem solving cannot be measured using the traditional quantitative techniques of educational research. Instead, we need detailed protocol research into how students actually go about solving problems. Without the knowledge that such research would provide, we do not know enough about problem solving to teach general problem-solving skills. The problem with teaching general problem-solving skills is summed up by Hayes (1980, p. 146).

> If we are to do evaluation of a problem-solving course well, we have to show that the students spontaneously use the skills we have taught them outside the confines of the classroom in which they were learned (Hayes, 1980).

Until we can show that this transfer of learning is possible, we will not be able to claim that it is possible to teach general problem-solving skills.

Other Literature Regarding Computers in Education

Far too many educators seem bedazzled by microcomputer technology. Witness, for instance, the professional literature where computers are often heralded in terms akin to a fundamentalist preacher praising the second coming. Words and phrases such as "fantastic," "greatest human advance since the printing press," "education's next great revolution" abound. The entire October, 1979, issue of Educational Technology typifies this reverence. Article after article sings the praises of the new technology, somehow assuming that microcomputers will improve education by their mere presence, and we had better hurry, hurry, hurry to get micros and computer literacy into the schools before it is too late. No article deals with the thought processes that underly what it is computer scientists do. None deals with the problem-solving skills that computer science can help build. It almost seems that some educators become so excited by computers that they do not want to think before they act for fear of finding out that all is not what they want it to be.

The frenzied praise can still be found in more recent articles but, in general, the rhetoric has grown more subdued. For instance, the August, 1983, issue of Popular Computing contains a thoughtful special report on computers

in education entitled "Computers: The Next Crisis in Education."

CHAPTER THREE

DESIGN OF THE STUDY

To construct a curriculum that would get to the heart of computer science problem-solving skills, the following model is proposed:

STEP 1: DEFINE PROBLEM-SOLVING SKILLS ESSENTIAL TO COMPUTER SCIENCE.

To discover the problem-solving skills essential to computer science, expert computer scientists were consulted. It was felt that the best place to find a high concentration of such consultants was a university computer science department. University scholars were felt to possess state-of-the-art knowledge over a wide range of computer science specialties.

The faculty members of the Computer and Information Science Department of the University of Oregon were each asked to submit a semi-structured interview (see Appendix A). The purpose of the interview was to discover what the faculty considered to be the essential problem-solving skills required by computer science. The University of Oregon's CIS faculty consists of approximately 15 full-time and part-time faculty. Of these, six of the full-time staff were interviewed.

Each faculty member was asked essentially the same

questions. A structured interview technique was chosen, instead of a questionnaire, because it permitted a particular question to be pursued to as great a depth as was necessary to obtain each faculty member's complete opinion. It was felt that other data gathering techniques did not allow for the depth of information that the author sought (Borg and Gall, p. 310). A structured interview also allows for more flexibility than a questionnaire.

The major weakness of any interview technique is that rapport between interviewer and subject might induce subjectivity and bias. To combat this tendency, each faculty member interviewed was asked the same questions in the same order.

A problem with interviewing experts in a technical field such as computer science is that the interviewer must be quite knowledgeable about the subject in order to conduct an intelligent interview. For instance, in a series of interviews like those conducted here, it would have been difficult for the interviewer to obtain worthwhile information without possessing a considerable amount of knowledge about both computer science and elementary education. The interviews in this study were conducted by an individual with this requisite knowledge. The interviewer had taken over 70 quarter hours of computer science courses, including 44 hours of graduate courses in computer science at the University of Oregon. In addition, the interviewer proposed a master's

degree in elementary education from the University of Arizona and ten year's experience as an elementary school teacher. This knowledge and experience enabled the interviewer to focus on the key element of this dissertation: the problem-solving skills from the discipline of computer science that can be taught to elementary school students.

After the interviews were conducted, they were analyzed to discover the main points of agreement and disagreement. If discrepancies were found, the involved faculty members were interviewed a second time. The goal was to distill from the interviews the key problem-solving skills that experts in the field felt were essential to the discipline of computer science.

STEP 2: DEVELOP ACTIVITIES SUITABLE FOR ELEMENTARY SCHOOL STUDENTS.

The activities developed for this dissertation were based upon the problem-solving skills defined in STEP 1. The goal was to develop activities that would present to elementary school students the problem-solving skills that the scholars interviewed in STEP 1 felt were the key problem-solving skills required by computer science.

Sources of activities included:

(1) Teachers Currently in the Classroom:

Many elementary school teachers are already using computers in their classrooms,

and they were generally willing to share with the curriculum developer the activities that have worked best for them. The curriculum developer could then decide whether these activities could be fit into one of the areas defined in STEP 1.

Classroom teachers are very resistent to curriculum imposed on them from above. To avoid such problems, it is essential to involve classroom personnel in the curriculum process as early as possible. Early involvement also takes fullest advantage of the considerable expertise possessed by teachers. They are education experts, and their opinions and ideas should be actively pursued in developing a unit such as that proposed here.

(2) Graduate Students in Computer Science Education:

If you have access to a university that trains graduate students in computer science education, take full advantage of the situation. Most students in this area have had extensive classroom experience to which they have added advanced training in computer

science education. Such people are a potential gold mine of useful information.

(3) The Literature:

Many popular journals and magazines, such as The Computing Teacher, Electronic Learning, Personal Computing, Creative Computing, and so on, contain articles on instructional computing. Occasionally, these articles contain an activity which could be of some help in preparing this dissertation.

(4) Elementary Textbooks, Usually in Mathematics:

Several elementary math text books were found to contain computer-related exercises. Most of these activities were related to flow-charting. Several other books of computer-related activities for elementary school children were found that yielded some useful activities.

(5) Activities Books Prepared by Teachers' Centers and Text Book Publishers:

At least one text book publisher, Houghton Mifflin (Houghton Mifflin, 1983) has published a booklet consisting of all of the computer-related activities from its elementary school mathematics series, Houghton Mifflin Mathematics.

> The WLOCNC (West Linn, Lake Oswego, Oregon City, North Clackamas School District) Computer Consortium (WLOCNC, 1982) published a looseleaf booklet to help the teachers in the district teach computer literacy. This booklet contains many computer-related activities, some of which might be suitable for a curriculum of the type developed in this dissertation.

The goal of this section is not to develop an entirely new set of activities never before seen in an elementary school classroom. Rather, the goal is to organize activities that already exist into a structure based upon the problem-solving skills defined in STEP 1. Only when an acceptable activity cannot be found from one of the sources listed above will a new activity be created.

STEP 3: FIELD TEST THE ACTIVITIES IN ELEMENTARY SCHOOL CLASSROOMS.

The purpose of field testing the activities was to get input from the students themselves. The goal was to eliminate as many "bugs" as possible from the activities before promulgating them as suitable for general classroom use. The activities that had been developed in STEP 2 were each tried out on a number of different types of students in a number of different school settings.

The first tryout of each activity was with a small group of three or four students in a room separated from their classmates. This isolation from other students was felt necessary to minimize distractions. Each activity was refined this way until it was deemed teachable enough to present to a larger group. Some of the activities were then taught to a group of at least 15 students, sometimes with the students' regular teacher observing. The observing teachers were asked for their evaluations of the content of the demonstration lessons, not for their evaluations of the demonstrating teacher's technique. More detailed teacher input was sought in STEP 5.

Only some activities were taught to entire classes because the goal of this dissertation was to develop a model for developing a curriculum. The goal was not to develop a methodology for teaching computer science education.

The unit being developed by the author will last at least two or three weeks for approximately 45 minutes to an hour each day. It was felt that a unit of this length could be fit into the existing curriculum. A unit longer than this would require something else being eliminated from the existing curriculum.

STEP 4: INTERVIEW STUDENTS TO GET THEIR OPINIONS OF THE ACTIVITIES.

At the conclusion of each demonstration lesson (whether

to a small group or an entire classroom), each participating student was asked to complete a questionnaire about the activity (see Appendix B). In addition, a selected number of students was interviewed, using a structured interview technique.

It was felt that student input is vital to developing a curriculum that actually works in a classroom setting. No one knows more about this end of curriculum than do students, even if they are only nine to twelve years old. Students were interviewed before the activities were shown to teachers to avoid the interviewer being biased by the opinions of adult teachers, which were usually more eloquent and forceful than those of their students.

STEP 5: INTERVIEW TEACHERS TO OBTAIN THEIR OPINIONS ABOUT THE ACTIVITIES.

The activities were shown to a number of experienced classroom teachers who work with elementary school children. Included in those interviewed were each of the teachers whose classes I worked with.

Each teacher interviewed was asked to read the entire packet of activities (see pp. 66-108) before being interviewed. Each teacher interviewed was also asked to fill out a detailed questionnaire (see Appendix C).

STEP 6: DEVELOP AND PILOT TEST A WORKSHOP TO TRAIN A SELECTED GROUP OF TEACHERS TO USE THE ACTIVITIES DEVELOPED AND REFINED PREVIOUSLY.

A one-day workshop was developed to teach the techniques necessary to implement the activities developed in previous steps. At the conclusion of the workshop, each participant was asked to complete an evaluation (see Appendix D).

CHAPTER FOUR

IMPLEMENTING THE MODEL

This chapter provides a description of an implementation of the curriculum model introduced in Chapter Three.

Step 1: Define the Problem-Solving Skills Essential to Computer Science.

Each faculty member was asked to comment upon the importance he or she attached to a group of specific problem-solving skills that the interviewer felt were important to the successful pursuit of computer science. These skills included:

(1) General Intelligence

While not a problem-solving skill, this trait was included to see if those interviewed felt that computer science requires a level of intelligence so high as to preclude teaching computer science problem-solving skills to young children. If it takes too high a level of basic brains to study computer science, the attempt to teach these skills to young children should not be undertaken.

Specifically, this topic was included to

help discover the types of thought patterns used by computer scientists in their work.

(2) Persistence

Persistence was felt to be important in many areas of computer science, such as debugging and learning how to program. Problems encountered in computer science are often difficult to solve. They often yield solutions only after many hours have been spent struggling with them. Many false starts may be made, an occurrence which puts great emphasis on the ability to work persistently.

(3) Math Skills

Math Skills were included in the interview for the same reason that general intelligence was included: namely, to discover if the threshold level of math skills required to work in computer science was too high for many students in elementary school.

Mathematics has been closely allied with computer science for many years. Many universities, such as Rice Institute of Technology in Houston, Texas, still have not split off computer science into its own department. At Rice, computer science is taught under the Mathematical Sciences

Department (Rice University Catalog 1982-1983). At the secondary level, this relationship between mathematics and computer science also exists. For example, the first computer course offered by the Houston Independent School District was called Computer Math, under the auspices of the Mathematics Department. Currently, this course is the only computer-related course which the Texas Education Agency has approved for high school credit (TEA, Austin, Texas).

While computer science has close, natural ties with mathematics, it is apparent today that computers can do much more than mathematics.

This question seeks to explore how important computer scientists think the relationship between computer science and mathematics is. Specifically, is it possible for poorer math students to benefit from studying computer science or will their poor math skills preclude them from doing so?

(4) Debugging Skills

Debugging Skills are considered by some

to be an important problem-solving skill that working with computers can help teach. Seymour Papert, the developer of LOGO, states that:

> One of the mainstays of the LOGO environment is the cluster of concepts related to "bugs" and "debugging." One does not expect anything to work at the first try. One does not judge by standards like "right--you get a good grade" and "wrong--you get a bad grade." Rather, one asks the question: "How can I fix it?" (Papert, p. 101)

Debugging Skills were included in the interview to see if computer science professors agreed or disagreed with Papert's contention.

(5) Top-Down Design.

Top-Down Design is a problem-solving technique that appears in many introductory programming books. See, for instance, Moursand's Basic Programming for Computer Literacy (1979).

Top-Down Design is often presented as a powerful tool in the search for solutions of large, difficult problems. It was included in the interview to measure the degree of importance computer scientists place upon these techniques and to see how widely used they are in the field.

(6) Programming Ability

Programming Ability is obviously a problem-solving skill that some people who call themselves computer scientists must have. But, how important is the ability to program? Is it an essential problem-solving skill without which an understanding of computer science problem-solving skills is impossible? Or, is it a sideshow--of little importance to the deeper thought processes that computer science requires?

This question is particularly important to educators because programming requires a certain amount of hardware, the purchase of which can quickly consume scarce educational funds.

(7) Precision

Many of the things computer scientists do, such as programming, require a great deal of accuracy or precision. There is little doubt that precision is important in computer science but can it be taught? Is it a problem-solving skill that can be learned in an academic setting?

This question was included to see how important computer scientists think

precision is in their work. Specifically, how did they develop precise work habits?

(8) Algorithmic Thinking

Algorithmic Thinking refers to the process of looking systematically for a solution to a problem, rather than relying upon hit or miss techniques. The concept of algorithmic thinking is assumed to be significant but how important is it to computer scientists in their work? Is it a problem-solving skill that is widely used, or is it largely ignored by those working in the field?

(9) Teamwork

Many problems in computer science are too large to be handled by one person. This question was included in the interview to see the importance computer scientists place upon teamwork. Do they themselves work in teams while conducting research? Do they permit or require their students to work in teams while solving problems?

One of the advantages often stated for programming languages such as PASCAL is that they permit programs to be modularized. This process permits teams of programmers to

work together on different parts of the same project.

(10) Social Concerns

To what degree should computer scientists be concerned about the impact computers are making upon our society? This is a specific example of the general question: "How great a social concern should scientists have?"

This question was included to discover how computer scientists address this issue in their daily work.

Results of Interviewing Professors

A total of ten professors (six at the University of Oregon and four at the University of Houston) agreed to be interviewed. Each professor was asked the question in the order presented here. If a participant did not respond or did not elaborate beyond a simple yes or no answer, the response was not recorded.

(1) Intelligence: What level or type of intelligence does computer science require?

Each of the ten professors interviewed commented on this question. All but two felt that certain types of intelligence, or aptitudes, were required by computer science. These may be summarized as the ability to think

logically and directly. Their thoughts were expressed as follows:

> Computer science requires a structured sort of intelligence. A successful computer scientist must keep broad goals in mind while working on trivia. Goal trees must be kept in mind.
>
> A computer scientist must have an aptitude for math and logic.
>
> The ability to think abstractly is important, as is the ability to construct logical models out of generalities.

Only one respondent thought that intelligence was relatively unimportant. This professor felt:

> All students can handle computer science: computer science involves the same things as language.

This professor felt that "music training, symbol games, and reading and language skills" required the same or very similar skills as computer science.

Another respondent had trouble with the term computer scientist. He felt a distinction should be made between computer science and data processing, with the former requiring quantitative skills and the latter verbal reasoning skills.

Thus, most of the respondents felt that computer science does indeed require a certain type of intelligence, an intelligence that enables the individual to solve a problem logically and systematically. None felt that the level of intelligence required by computer science was so

high that only an exalted few could benefit from its study. Rather, a certain type of patterned thinking was required, and these patterns could be taught.

(2) Persistence: How important is persistence when solving problems in computer science?

Each of the ten respondents to this question felt that persistence was an important ingredient in the successful pursuit of computer science. Some of their comments follow:

> Persistence is important. The abilities to concentrate deeply and not give into distractions are important.
>
> Insights are usually made way ahead of where knowledge currently exists. Then hard, persistent work fills in the blanks.
>
> Often, bright people have no patience. In computer science, the goal is brightness plus persistence.
>
> Definitely important (three respondents).
>
> Persistence is the spark of creativity. It is especially important for doing successful research.

Of all the questions asked the professors, this question drew the most homogeneous responses. All of the professors questioned felt that persistence was a necessary component of problem solving in computer science.

Of particular importance was the fact that none of the respondents indicated that persistence could not be taught. They felt there was nothing inherent in persistence that precludes it from being instilled in all students, regardless of age or ability level.

(3) Math Skills: How important are math skills to solving problems in computer science?

The responses to this question were mixed. While those interviewed all responded, there was no general consensus in their replies. Some felt that math skills were an important prerequisite for their work:

> Math, especially algebra, is a computer scientist's bread and butter. Induction, deduction, and predicate calculus also are very important.
>
> Math, especially formal math, is important for a computer scientist.
>
> Math is really important for a computer scientist, especially logic and proof skills.

Some of the respondents took a different position:

> Math is important for all science because the language that is best for describing phenomena. But, it is not always necessary for success in programming or computer science.
>
> For theoretical work and some other areas of computer science, math is important. But, for other areas of computer science--such as data processing--a high level of math skills is not necessary. Analytical ability is important.
>
> The importance of math skills depends upon where you do your work. In some areas of computer science, math is necessary. Other computer scientists almost never use math.

Some of the respondents felt that math was vitally important to computer science. Each of the other respondents at least felt that math was important in some

areas of computer science. Thus, it may be concluded that a certain level of math skills is a necessary prerequisite to problem-solving in computer science.

It is interesting to note that those respondents who claimed that math was vitally important all did their work in an area of computer science where math was indeed very important: algorithm analysis, artificial intelligence, or vision.

(4) Debugging Skills: When solving a problem in computer science, how important is it to be able to debug something that does not work like it should--a program, a proof, etc?

All ten professors in the sample responded to this question. Each of the respondents felt that debugging skills were important for the successful pursuit of problem solving in computer science. Some of their comments were as follows:

> The concept of debugging is important, especially in programming. Debugging is best taught by example. To learn how to debug an error, the student should listen to someone dissect a problem.
>
> Deductive skills are an important part of debugging.
>
> The more you know about a subject, the better you will be at debugging.
>
> Debugging is closely tied to persistence. You cannot quit easily and be good at debugging.

One of the respondents made the interesting observation that debugging is a skill that is not limited to computer science:

> For example, paragraphs can be looked at as programs which must be debugged many times to get them just right.

To summarize: the ability to debug was seen as an important problem-solving skill in computer science by all of those interviewed.

(5) Top-Down-Design: How important are the principles of top-down-design when solving problems in computer science?

The responses to this question addressed two basic points: the importance of top-down-design as a problem-solving tool and the difficulty involved in teaching the concept. While most of the respondents felt that top-down-design was important, there was a great deal of confusion over when and where to teach it.

> Top-down-design is critically important in computer science, especially when debugging.
>
> Top-down-design is important, but the problem is how to teach it. Maybe by working through examples with the class?
>
> It is very important but very difficult to get students to implement.
>
> Top-down-design is an important principle of systems analysis.
>
> Top-down-design is good for large projects. All large data processing projects need to use it.

> It should not be taught at the introductory level.
>
> It is important, but too often students are taught to use pseudo code, and they do not understand why it is needed.
>
> Top-down-design is implicit in math, business, or any large-scale task in almost any field. If the problem is too easy, top-down-design will not be necessary, but the concept should be taught in beginning courses.

One of the conclusions which emerged was that top-down-design was most useful for large, complex problems, but that it was often taught to beginning computer science students who were dealing with relatively simple problems.

When deciding whether or not top-down-design skills are teachable to elementary school children, these problems of how to teach it will be magnified. It will be difficult to find or create activities at the correct level of difficulty. The problems must be difficult enough so that the solutions are not trivial, while at the same time not so difficult as to cause undue frustration.

(6) Programming Ability: How important a problem-solving skill is programming ability in computer science?

While there were some respondents who felt that programming ability was not vitally important, most felt programming ability was an important problem-solving skill in computer science.

> The important concept to garner from programming is an appreciation that there are lots of recipes for the same cake. This is a critical issue in computer science.
>
> A student should develop an appreciation for a well-written program, much as he or she should learn to appreciate good music. But, not everyone needs to be a virtuoso.
>
> You do not have to be a hacker, but a computer scientist should be reasonably competent at programming.
>
> An important concept to get out of programming is that right answers are not all that important.
>
> Programming helps students develop thinking skills at the algorithmic level.
>
> Programming teaches one how to implement logical design. But, too much programming can hinder a designer or abstract thinker.
>
> Programming has many facets: for example, efficiency is more important than speed.

Only one respondent felt that programming ability was an extremely important problem-solving skill in computer science.

> Programming is absolutely important. An individual must be a fluent programmer to be a good computer scientist. You must know the rules and how to apply them before you can break them and discover something new. And, programming is one of the "rules" a computer scientist needs to know.

Similarly, only one respondent took the opposing view--that programming ability was not necessary for doing quality work in computer science.

> Programming ability is not important at all to understanding computer science.

It is interesting to note that neither of the two respondents expressing these extreme views was trained originally as a computer scientist. One was a biologist, the other a businessman.

The general conclusion these responses lead to is that some programming ability is an important problem-solving tool in computer science, not simply to solve problems by writing programs but to gain an understanding of the type of thinking that is important to solving other types of problems in computer science.

Implementing programming in an elementary school curriculum will present some problems, such as which language to teach and how to train teachers to teach programming.

(7) Precision: How important a problem-solving skill in computer science is precision?

Two-thirds of those interviewed responded to this question. Not surprisingly, each of those responding felt that precision was an important component of problem solving in computer science.

> Precision is a prerequisite for doing quality work in computer science.
>
> If you are not precise, you will get nowhere. Either your colleagues or the computer will not tolerate imprecision.
>
> Precision is quite important, especially when communicating with non-computer scientists.

The general consensus of these opinions is that precision is an important problem-solving tool in all phases of computer science.

(8) Algorithmic Thinking: How important a problem-solving skill is algorithmic thinking?

Only half of those interviewed responded to this question. The answers were mixed--with some feeling algorithmic thinking important and others feeling the opposite.

> Algorithmic thinking helps clarify a problem. It can provide a structure for building a solution.
>
> Sometimes too much attention to algorithmic thinking hinders finding a solution. Too much time is spent looking for an algorithm and not enough time looking for an answer.
>
> Algorithmic thinking is as much a hindrance to problem solving as it is a help.
>
> For kids, algorithmic thinking should be taught as developing plans. Kids should be allowed to solve problems--such as how can the class have a picnic tomorrow--activities that require planning. They should work out the solution with one another by talking things over. They should develop an attitude of 'Here's a problem, now let's solve it'.

While it is difficult to develop a consensus from these responses, the last response seems the most useful for an elementary school setting. Children can be given experience in solving complex problems that make sense to them. These

problems should deal with topics that do not expose them to dangers that expose them to excessive adult supervision. The students should be trained to sink or swim with their decisions.

(9) Teamwork: How important for problem solving in computer science is the ability to work well in teams?

Each of those interviewed responded to this question. Most felt that the ability to work in teams was important in some areas of computer science but unimportant in others.

> Teamwork is less important for a researcher than it is for someone working in industry.
>
> Some researchers do good work working alone. Berkeley's UNIX group, on the other hand, must work as a team. Some jobs are just too big for one person.
>
> Teamwork is not terribly important in computer science. It should be learned in gym--not in computer science.
>
> Teamwork should be stressed early on in the curriculum. Peer review of ideas and opinions should start early.
>
> Teamwork is an important concept. A great deal of top quality work in computer science has been done in teams. For instance, much of Newell and Simon's work was done in a team.

The general consensus is that teamwork is often an ingredient in the completion of successful work in computer science. None of the respondents thought it hindered quality work, although some felt it was not always necessary.

It should be relatively easy to put activities that stress teamwork into the elementary school curriculum. One possible obstacle may be the traditional method of evaluation used by educational institutions in this country, with the emphasis on evaluating students as individuals rather than as part of a team.

(10) Concerns for Social Impact of Computers: How important is it for a computer scientist to be concerned with the social implications of his or her work?

All of the respondents commented upon this question. The consensus was that the issue was important but difficult to teach.

> The stock answer is that a social conscience is important, but the difficult issue is how do you teach it?
>
> More discussion of the social impact of computers is needed in pre-college education.
>
> Kids are way ahead of adults in this area because they have an intuitive understanding of the technology. They do not fear computers.
>
> Social concern should be taught, but how? A lot of what goes on is b.s.
>
> You can ignore the issue and still be successful, but you should be concerned.
>
> One example of an issue that should be addressed is the fact that about 80 percent of AI research is funded by the military.

> Kids should be made to realize that their world is much different than their parents' world.

The consensus is that some mention of social responsibility should be made in pre-college education. The problem is how to include it in the elementary school curriculum.

(11) Miscellaneous Comments

Each computer scientist who was interviewed was asked to comment upon any area of computer science education that he or she wanted. These comments covered a wide range of topics.

> Computer science is important for kids to learn. They are interested, and they can learn quite easily at a young age.

> We are too hung up on computer science. We should be more concerned with teaching about computer tools, such as electronic spreadsheets, record management systems, and word processing.

> At the elementary level, there is no need for every child to study computer science. It should be an elective, like woodshop.

> LOGO seems to be a good way to go with young children. Young children have short attention spans, and LOGO can help hold their attention.

> Educational games should be stressed.

Conclusions

The primary problem-solving skills of computer science gleaned from interviewing the professors were:

(1) Algorithmic Thinking: The ability to solve problems in a logical and systematic way.

(2) Persistence: The ability to work persistently.

(3) Debugging: The ability to find and correct errors.

(4) Precision: The ability to work precisely.

It was the consensus of those interviewed that each of these skills is an important component of problem-solving in computer science.

In addition, there are three skills that were mentioned often enough to be considered important enough to include in any curriculum. These include:

(5) Top-down-design: The ability to break down large problems into smaller, more easily solved components.

(6) Programming Ability: The ability to write, modify, and debug simple programs in a high level language.

(7) Teamwork: The ability to work together with other people to solve problems.

Each of the activities described in STEP 3 of this dissertation concentrates on one or more of these problem-solving skills.

There is no reason why these problem-solving skills cannot be introduced into the elementary school

curriculum. These seven skills are meant to be part of a methodology of approaching problem solving in computer science. The methodology represented by these seven skills is seen as more important than the content of the activities. Activities to sharpen the application of this methodology could be developed at any level, from elementary school to university. What is important is that the problem-solving skills listed above were identified via a systematic method of interviewing experts in the field of computer science.

Step 2: Develop Activities Suitable for Elementary School Students.

The aim of STEP 2 is to develop activities that reinforce the problem-solving skills defined in STEP 1. As mentioned in Chapter Three, suitable activities may be obtained from a number of sources. The key point is that they must fit into the framework of problem-solving skills developed in STEP 1. It is not important that these activities be original, as long as they fit into the model. The specific sources used here included:

(1) Teachers currently in the classroom.

In the Spring of 1982, I observed Karen Beisse, a teacher in the Eugene, Oregon, Public Schools, teach a two-unit on computers to a class of fourth, fifth, and sixth graders. Each lesson was approximately one-hour in length.

Karen had developed a number of activities for the unit on computers. Some of the ideas had come from magazines, while some had been developed by Karen herself. She had used most of the activities for several years and, in many cases, she did not remember their origin.

It is helpful when designing a curriculum to observe experienced teachers teach, especially one as talented as Karen Beisse. The observer gets the benefit of seeing how a particular activity actually works in a real classroom situation. In addition, the observer gets the benefit of

the experienced teacher's ability as a curriculum designer. A dedicated teacher will not teach something she or he believes to be of little or no educational value. When an observer watches such a teacher, the observer gets to watch an activity a professional educator felt worthy of presenting to a class.

(2) Graduate students in computer science education.

The University of Oregon is noted for, among other things, its high concentration of computer science education students. Each summer, Professor David Moursund oversees an institute devoted primarily to the formal training of computer science educators. During the summer session of 1983, over 15 courses were offered educators interested in various aspects of computer science education (University of Oregon Summer School Catalog, 1983).

The University of Oregon is one of the few institutions of higher education offering a Ph.D. in Computer Science Education. Many students who attend the summer session eventually enter the doctoral program. They come from all regions of the United States and several foreign countries. As of the Fall, 1983, there were 15 computer science education doctoral students at the University.

What makes the program at Oregon so valuable is the fact that each doctoral student must take a series of courses in both the College of Education and the Computer and Information Science Department. This combination of

of courses ensures that the program of study leads to a true computer science education degree.

While developing the ideas for this dissertation, the author was able to test his ideas on a number of fellow students of computer science education. These included University of Oregon students Graham Ferres, John Keating, and Richard Robbat.

(3) Computer science educators.

The Houston Independent School District's Department of Technology has a staff of approximately 30 professionals who work in various areas of computer science education. These areas include: training, planning, maintenance, software development, future studies, curriculum development, and research. Collectively, they have many hundreds of years teaching experience and many hundreds of hours formal course-work in computer science education. The staff includes six Ph.D.'s and an equal number of students who are currently working on their dissertations.

The Department of Technology also has a great deal of hardware and software for the staff to use and evaluate. Nearly every employee has an assigned microcomputer for use at work. Approximately 50 microcomputers at the Department's campus are available for use by teachers and administrators from the Houston Independent School District and visitors to the facility. The media center owns over 1,000 pieces of educational software and a large number of

books on various topics related to computer science education. Over 50 professional magazines and journals are received each month.

The sum total of these resources is a facility filled with knowledgeable people with state-of-the-art ideas for and about computer science education. Many of these professionals were more than willing to share their ideas with the author. They read critically ideas gathered from other sources and offered some ideas of their own, many of which have been incorporated into the activities used in STEP 3.

The Department of Technology and the University of Oregon have several important characteristics in common:

(1) A belief that computer science education is a matter to be taken seriously;

(2) A critical mass of people interested in the same professional pursuits;

(3) A large number of people with whom to interact; and,

(4) Close ties to both education and computer science.

These resources have been of considerable help in preparing this dissertation.

(4) The literature.

There have been numerous articles and books written about teaching computers and computer science to children.

Many of these articles and books contain activities that may be used with elementary school students. An example is Computer Tutor by Sandra Markle and Bev Armstrong (Markle).

When developing a curriculum, it would be deleterious to the quality of the final product to ignore the work of others in the field. Many people have already developed many good activities, and it is folly to believe that one person can always develop better activities from scratch. The collective creativity of these educators should not be wasted.

The weakness of much of this work is not the educational value of the individual activities but the lack of a structure within which the individual activity fits. One of the purposes of this dissertation is to propose a model for developing such a structure.

(5) Develop original activities.

When a suitable activity could not be found for a particular problem-solving skill. the author developed his own. The development process runs the risk of producing activities that will be flawed due to lack of previous testing. Sometimes this is a minor problem. For example, a flowchart for outputting the even numbers from 2 to 20 can easily be transformed from a pencil and paper activity into a programming activity by having students translate the flowchart into BASIC, or some other suitable language. At other times, there is no simple solution as, for example, when an activity was sought to give students practice with

For this topic, the author had to develop the entire activity from scratch.

In conclusion, there are many sources of activities the author had available to him when developing this dissertation. The above list is not all inclusive, but each was in fact used by the author. In addition, the author's 10 years of elementary school teaching experience and his training in computer science were also important. Most of the activities were stimulated by one of the above sources and then modified by the author.

Step 3: Field Test the Activities.

Activity: Predicting Outcomes

Purpose: To give students practice with:

(1) Tracing through the steps of an algorithm;

(2) Predicting an algorithm's output; and

(3) Modifying an algorithm.

Specific Skills:

Algorithmic thinking; debugging; programming skills.

Student Prerequisites:

Students should know what an algorithm is; programming ability (if the programming option is chosen).

Teacher Prerequisites:

Programming ability (if the programming option is chosen).

Materials:

Ditto masters; microcomputer(s) (if the programming option is chosen).

Time Required:

One or two class periods.

Source:

Author

Directions: Follow the steps of this algorithm. Write down whatever the algorithm says to write. Keep going until you reach STOP.

Step 1: Write 0.

Step 2: Add 2 to the number you have just written.

Step 3: Write the sum you have just obtained.

Step 4: Keep repeating Steps 2 and 3 until you write 20.

Step 5: Stop.

Questions to discuss with class:

(1) What would happen if Step 4 were changed to: Keep repeating Steps 2 and 3?

(2) What changes would you have to make in the algorithm so that all the odd numbers from 1 to 19 would be written?

(3) What changes would you have to make in the algorithm so that all the even numbers from 2 to 100 would be written?

Programming Option

The above activities can easily be programmed in BASIC for an Apple or other microcomputer. For example, the first activity becomes:

```
10  LET SUM = 0
20  LET SUM = SUM   2
30  PRINT SUM
40  IF SUM = 20 THEN GOTO 60
50  GOTO 20
60  END
```

(1) What would happen if line 40 were eliminated?

(2) What changes would you make to print out the odd numbers from 1 to 19?

(3) What changes would you make to print out all even numbers between 2 and 100?

This activity was tested with two different groups of students, a small group of four fifth grade students from a regular class at Westmoreland School, Eugene Oregon, and an entire class of fifth grade students, also at Westmoreland. The author taught both groups.

It was felt this exercise would promote algorithmic thinking skills by helping students understand how a

particular algorithm worked. It would foster debugging skills by giving students practice in correcting algorithms that do not obtain the desired results and having students find out why.

Activity: Social Concerns

Purpose: To expose students to some of the social implications of the computer revolution.

Specific Skills:

Social concerns.

Student Prerequisites:

None.

Teacher Prerequisites:

None.

Materials:

None.

Time Required:

One or two class periods.

Source:

Author

Directions: Discuss the following questions with your students.

(1) Do you know anyone who works with computers?

(2) Do you know anyone whose job has been taken over by a computer or robot?

(3) What should people do if their jobs are taken over by computers?

(4) What if there are not enough jobs to go around?

(5) How would you feel if your job were taken over by a computer?

(6) Do you think computers will some day be able to do whatever a human being can do?

(7) What are some of the things you do today that you think computers will be able to do in the future?

(8) What will people do with all the free time computers and robots will provide for them in the future?

(9) What is the best thing computers will do for human beings in the future?

(10) Do computers scare you in any way? How?

The author taught this activity to a group of four fifth graders and an entire fifth grade class at Westmoreland Elementary School in Eugene, Oregon.

The goal was to help students understand their own feelings and thoughts about computers and to get them to think about computers in both a positive and a negative light.

Activity: Following Directions

Purpose: To illustrate the importance of precision.

Specific Skills:

Precision.

Student Prerequisites:

None.

Teacher Prerequisites:

None.

Materials:

None.

Time Required:

One-half class period.

Source:

Author

Directions: Tell your students to take out a pencil and paper and do exactly as they are told. Read the following eight directions aloud to them.

(1) There will be no talking!!! Please listen carefully and do exactly as you are told.

(2) Don't do anything until I say so (please use these exact words!!!).

(3) Ready? Let's begin. Put your name on the top line of the paper, beginning at the left margin.

(4) Number your paper from 1 to 10 down the right margin starting at line 5 and skipping lines.

(5) Circle number 4.

(6) Underline your last name.

(7) On line 6, spell CAT backwards.

(8) Put down your pencils.

Ask the students what they have written on their papers. Now the crusher. Tell them they should not have anything written on their papers. That's right, no one should have written anything. Why? Because you didn't say so (see number 2 above.

Discussion

Ask your students why they wrote what they did. Was it because they can think? Point out that computers cannot think, at least not yet. A computer would not have done anything until you said so. This is an important point to understand about computers as they presently exist. Computers cannot figure out what the programmer means. The programmer must tell the computer exactly what to do, and exactly means just that--exactly. People are not like that, at least not yet.

Directions: Follow these directions as closely as possible. If a direction tells you to do something that another direction tells you not to do, follow the direction given first.

(1) Take out a piece of lined notebook paper.

(2) Write your name on the top line starting at the left margin.

(3) Write today's date on the line immediately following your name.

(4) Do not do what lines 5, 6, 7, and 8 tell you to do.

(5) Write "HELLO" on line 7.

(6) Circle your name on line 1.

(7) Underline the date.

(8) Put an "X" through the "HELLO" you wrote on line 7.

(9) Circle the date.

(10) Stop.

This activity was tested on a group of four fifth graders at Westmoreland Elementary School and a group of four fourth graders at McCornack School, both in Eugene, Oregon.

These exercises were chosen because they stress a patient approach to detail that requires a certain precision on behalr of the students.

Activity: Spiffy the Spider

Purpose: To provide students with a programming experience.

Specific Skills:

Algorithmic thinking; precision.

Student Prerequisites:

The abilities to recognize numbers 1-8 and letters A-H.

Teacher Prerequisites:

An understanding of a coordinate system.

Materials:

An overhead projector would help; dittos of the gird; an Apple IIe or II Plus microcomputer.

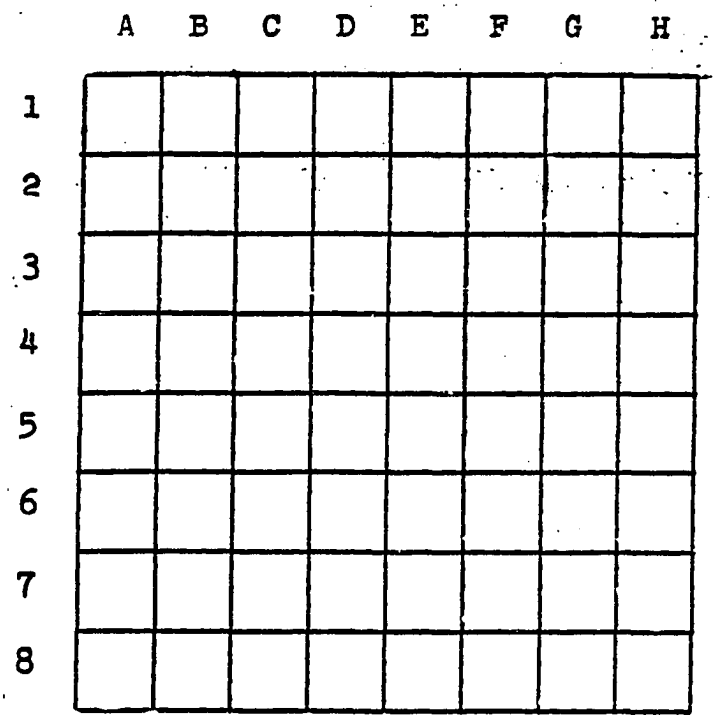
Time Required:

One or two class periods.

Source:

Karen Beisse (1982)

"Spiffy" is a spider that can be moved around on an 8 x 8 grid, such as a checkerboard. The grid looks like this:



"Spiffy" only understands four simple commands:

U - Up one square

D - Down one square

L - Left one square

R - Right one square

Directions: The general idea behind "Spiffy" is to have students write programs to move the spider around the grid. For example, if "Spiffy" is at location 8A, the following sequence of steps (a "program") would move it to location 6E: R, R, R, U, U, R. Note that any combination of four R's and two U's will accomplish the task. This fact is a fairly important point for students to learn about

programming. There is usually more than one way to accomplish the same task. You could set a destination for "Spiffy" and have students discover the shortest possible route. In the following example, the shortest route involves six steps, but it might take a student a while to see that it is impossible to move from 8A to 6E in less than six moves.

Variations

(1) Put obstacles on the grid by darkening certain squares and have students move "Spiffy" around between two points avoiding the obstacles.

(2) Use a checker board and a checker to simulate "Spiffy" and the grid.

(3) Give "Spiffy" another command: C for Color. When "Spiffy" is given the instruction "C", the box it is in gets colored in. Now "Spiffy" can make designs!

(4) Put numbers in each of the 64 squares on the grid and have "Spiffy" move around the grid and sum the numbers of the squares he passes through. For example, have the students program "Spiffy" so that the sum of each box passed through totals a particular number, say 106. Is there more than one trail that will accomplish this task? Which path will

reach the desired sum in the fewest number of steps?

(5) Put a different letter in each of the 64 squares on the grid and have "Spiffy" spell words. "Spiffy" may need a new command, R for Read, to accomplish this task. When given the command "S", "Spiffy" will read whatever letter is in the box he is currently on and add it to the characters already read.

This activity was taught to a group of four fifth grade students and an entire fifth grade class at Westmoreland Elementary School in Eugene, Oregon.

It is intended to give students a high-interest activity that requires pre-planning and precision in thought.

Activity: Interpreting Algorithms

Purpose: To give students practice writing and debugging algorithms.

Specific Skills:

Algorithmic thinking; debugging.

Student Prerequisites:

Some knowledge of algorithms; map reading skills.

Teacher Prerequisites:

None.

Materials:

Transparency of a map; overhead projector.

Time Required:

One class period.

Directions:

Write a series of directions to go from:

(1) Pharmacy to library

(2) Park to Police headquarters

(3) Shopping mall to doctor's pavillion

Variations:

Debug these directions for going from the hospital to the book store.

(1) Walk out the hospital onto Main Street

(2) Turn left

(3) Turn left on Calhoun Street

(4) Enter the book store

Source:

Reston and Hunter (1983)

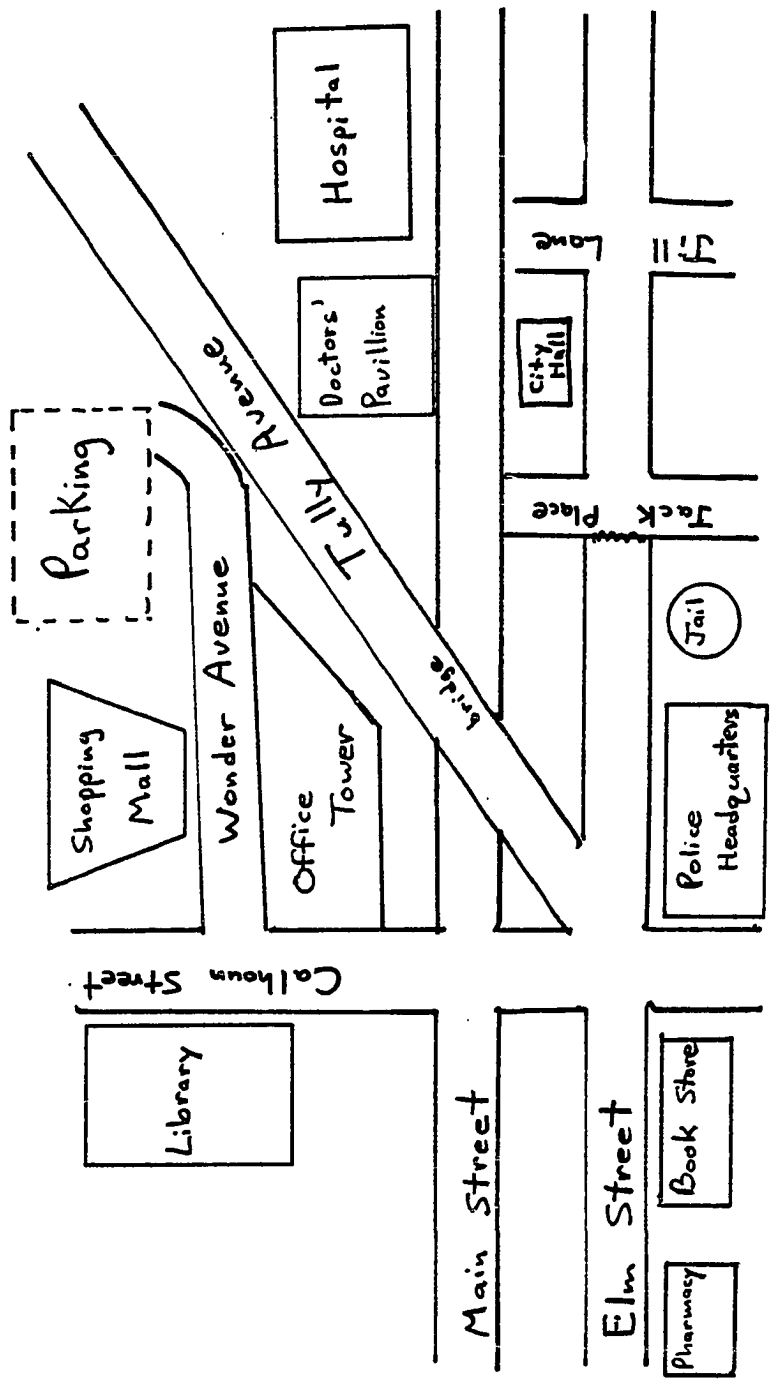


Figure 1. Map for Exercise.

This activity was taught to a class of approximately 25 fifth graders at Westmoreland Elementary School in Eugene, Oregon. The activity was developed by the author after a discussion with fellow graduate students Graham Ferris, Dick Robbat, and John Keating in which we concluded that computer science problem-solving skills could be incorporated into other subject areas. Here this other subject area is map reading.

Activity: Tic-Tac-Toe Algorithm

Purpose: To give students practice following the step-by-step sequence of an algorithm.

Specific Skills:

Algorithmic thinking; precision.

Student Prerequisites:

Knowledge of how to play Tic-Tac-Toe.

Teacher Prerequisites:

Knowledge of how to play Tic-Tac-Toe.

Materials:

Blackboard or ditto master upon which to put a Tic-Tac-Toe grid.

Time Required:

One class period.

Source:

Moursund; Introduction to Computers in

Education for Elementary and Middle School Teachers (1981, p.37).

Karen Beisse (1982)

Here is an algorithm for Tic-Tac-Toe that is difficult to beat if the person who goes first follows the steps outlined. Note that you might not win but, if you follow the algorithm, you will at least be difficult to beat.

Begin by numbering a Tic-Tac-Toe grid on the blackboard as follows:

Place an "X" in any of the nine squares. The algorithm is written so that "X" almost never loses if "X" goes first. If you are partial to "O", simply change the "X's" to "O's" in the algorithm that follows. The algorithm: after the first move, each time it is your turn, take the lowest numbered move of the four listed below that is available to you. Note: A file is any sequence of three consecutive squares. A file may be vertical, horizontal, or diagonal.

(1) If a file contains two "X's" and no "O's", put an "X" in the empty square of the file. If two such files exist, play the square with the lowest number.

(2) If there is a file with two "O's" and no "X's", put an "X" in the empty square in that file.

(3) If possible, make two files with two "X's" and no "O's". If more than one such move exists, play the lowest numbered square that gives the desired results.

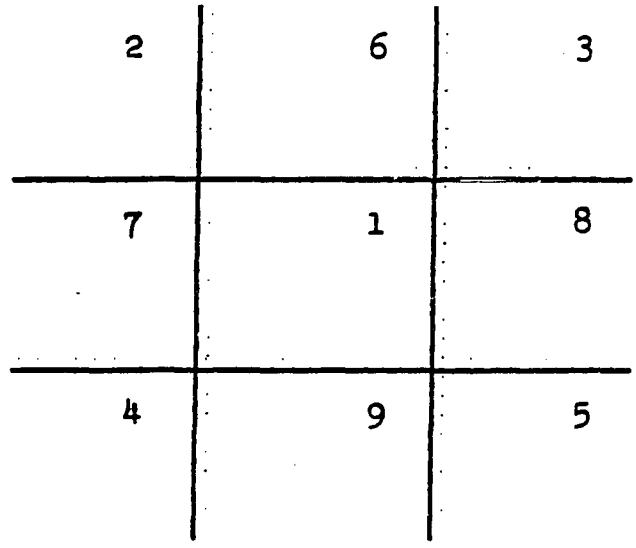
(4) Play the unused square with the lowest number.

Variations and Questions

(1) Let the player not using the algorithm go first. Will the algorithm still avoid defeat every time? (No)

(2) Play the second move available instead of the first. Will the algorithm avoid the agony of defeat every time? (No)

(3) Try different numbering sequences for the squares. Will the algorithm remain invincible? (Sometimes Yes, sometimes No)



This activity was tested on a group of four fifth grade students and an entire fith grade class at Westmoreland Elementary School in Eugene, Oregon. The author taught both sessions.

These activities were motivated by observing Karen Beisse teach the lesson to her class of fourth, fifth, and sixth grade students at Evergreen Elementary School in Eugene.

The goal is to give students practice in precision by having them trace through an algorithm that demands attention to detail.

Activity: Algorithm Development

Purpose: To increase a student's ability to solve a problem in an algorithmic, or step-by-step manner.

Specific Skills:

Algorithmic thinking; teamwork; top-down-design.

Student Prerequisites:

None.

Teacher Prerequisites:

None.

Materials:

Peanut butter and jelly (if you choose to actually make the sandwiches).

Time Required:

One class period.

Source:

Karen Beisse (1982)

Markle and Anderson (1981)

Directions: Have each student write out a sequence of steps that might be followed by someone preparing a peanut butter and jelly sandwich. One possible answer:

(1) Check to see if you have peanut butter, jelly, and bread. If you do not, go buy them.

(2) Take out peanut butter, jelly, and two slices of bread.

(3) Get knife.

(4) Spread peanut butter on one side of one of the slices of bread.

(5) Spread jelly on one side of the second slice of bread.

(6) Put two slices of bread together so that peanut butter and jelly sides go together.

(7) Clean up mess!

There are many possible answers, each correct in its own way. Have the students share their answers with the class.

Questions to discuss with class:

(1) Which steps could be put in different places in the sequence without affecting the final outcome? (For example, in the above answer, Steps 2 and 3 could be interchanged without changing the results.)

(2) What would happen if Steps 2 and 5 were interchanged?

(3) What could go wrong if Step 1 were omitted?

(4) What could go wrong if Step 6 read as follows: Put the two slices together (sandwich might have spreads on the outside).

Variations

(1) Divide the students into teams of four or five.

(2) Have each team write a sequence of steps for preparing a peanut butter and jelly sandwich.

(3) Have each team exchange its plans with another team.

(4) Provide peanut butter, jelly, bread, knives, napkins, and so forth, and instruct each team to follow the other team's directions for making a sandwich.

Discuss with class:

Would the results be different if the steps were placed in a different sequence? Have the students identify the critical steps in the other team's algorithm. (A critical step is one that cannot be placed in a different position without changing the outcome.)

Have the students, working in teams of three or four, write out an algorithm for building a house. This problem is far too complex for children to answer completely. That is part of its value. Many real life programming projects are far too complex to be solved easily.

Some other ideas for algorithms for student teams to work on include:

(1) Traveling to and from school.

(2) Planning a field trip.

(3) Following a recipe from a cookbook.

(4) Making change (can be tested with real coins).

This activity was tested on a group of four fifth grade Pace students at Westmoreland Elementary School in Eugene, Oregon, and on an entire class of fifth grade students at the same school. The variation in which the students were making the sandwich was not done with the entire class. The activity was designed to give students experience with designing algorithms, working in teams, and performing top-down-design. The last two variations are especially designed for top-down-design because they are too difficult to solve quickly.

Activity: Hierarchies

Purpose: To give students an experience with a math hierarchy commonly used in computer languages.

Specific Skills:

Algorithmic thinking; persistence; debugging.

Student Prerequisites:

Students should understand basic arithmetic facts (+, -, *, /).

Teacher Prerequisites:

Basic arithmetic skills.

Materials:

Chalkboard on which to work demonstration problems.

Time Required:

One or two class periods.

Source:

Author

How does a computer calculate an expression such as 9 + 3 x 3? We know it can calculate 9 + 3. We know it can calculate 3 x 3. The problem is this: would the 9 + 3 be performed first with the resulting sum (12) multiplied by 3 (giving a final value of 36), or would the 3 x 3 be performed first with the resulting product (9) added to 9 (giving 18)?

To solve this problem, the developers of computers elected to use the rules of algebra. Multiplication and division are performed first, addition and subtraction second. This information may be summarized in the following hierarchy:

x, /.	Performed first.
+, -	Performed after all multiplications have been carried out.

Following these rules, the above example would evaluate to 18. The multiplication is performed first, yielding 9. This product is added to the 9 giving the final answer of 18.

Other examples:

7 x 2 - 2 = 12 (7 x 2 first, then subtract 2)

7 - 2 x 2 = 3 (2 x 2 first, then subtract 4 from 7)

Another problem arises when two operators at the same hierarchy level occur in the same expression, as in:

8 / 4 x 2

Which operation is performed first, the / or the x? The following rule governs such situations: When two operators have the same hierarchy level, they are performed in order from left to right. Thus,

8 / 4 x 2 = 4 (8/4 = 2, then 2 x 2 = 4)

Putting both sets of rules together gives the following <u>super</u> rules:

(1) Perform all / and x first in order from left to right.

(2) Perform all + and - next in order from left to right.

Here are some examples illustrating the application of these two rules:

9 + 3 / 3	(Answer: 10)
10 / 5 x 2	(Answer: 4)
12 + 6 - 2 x 9	(Answer: 0)
15 + 2 - 6 * 2	(Answer: 5)
8/4 + 4	(Answer: 6)

What, if anything, is wrong with the solutions to these problems?

8/4 - 2 = 4

6 * 3 / 3 = 6

4 + 3 * 8 / 4 = 14

This activity was taught by the author to two different groups: a group of five regular fifth grade students at Westmoreland and a group of 15 Pace students at McCornack Elementary School, both located in Eugene, Oregon. Both of the sessions took place in the media center.

This activity was developed by the author. The goal was to teach precision skills by requiring students to interpret each problem with care and precisely apply the rules to arrive at an acceptable solution. If the rules were not followed exactly, the wrong solution would result. Arithmetic hierarchies were also chosen because it is the way computers do arithmetic.

Activity: Student Robots

Purpose: To give students an experience writing and debugging algorithms.

Specific Skills:

Algorithmic thinking; debugging; teamwork.

Student Prerequisites:

None.

Teacher Prerequisites:

None.

Materials:

Chart or chalkboard upon which to display the commands chosen to direct the student robots.

Time Required:

One or two class periods.

Source:

Karen Beisse (1982)

Reston and Hunter (1983)

Directions: Have your students write programs with other students and have them try to act them act out. Limit the commands that each STROB (i.e., student robot) can understand to a relatively small number. The following is one possible set of instructions:

SU - Stand up

SD - Sit down

R_ - Turn right ___ degrees

L_ - Turn left ___ degrees

F_ - Go forward ___ steps

B_ - Go backward ___ steps

LA - Lift arms overhead

DA - Drop arms to side

SM - Say "Mama"

ST - Stop

The idea is for each student to write out a 10 or 20 line set of instructions, give this "program" to another student, and see if the STROB does what it is supposed to do. If an instruction is given that is not on the list of allowable instructions, the STROB should stop. The "programmer" should then try to debug the program to get the STROB to do what the programmer wants it to do.

Variations

(1) Set specific goals (for example, move a STROB to a particular corner of the room) and see who can write the most efficient program to accomplish this goal. The most efficient program is the one that accomplishes the task in the fewest number of instructions.

(2) Give the students a program that contains a "bug" and have them debug it. One such program:

Here is a program to move a STROB in a square. Debug it.

```
SU
F5
L90
F5
R90
F5
L90
F5
ST
```

After students have developed a program that works the way they want it to work, see if they can write another program to accomplish the same task.

Student robots was taught to a small group of four fifth graders at Westmoreland School, an entire fifth grade class at Westmoreland, and a fourth grade Pace class at McCornack Elementary School.

It was felt that STROBS was a good way to give students experience with several important skills. To write a workable "program" requires careful attention to algorithmic development. Debugging is encountered when errors are found, and the debugging is fairly easy to do because the student "programmers" can act out the correct pattern. Finally, teamwork is a natural occurrence when this activity

is used. Students must work together, if only because there must be a student robot and a student programmer.

Activity: Tracing an Algorithm

Purpose: To give students practice in tracing an algorithm.

Specific Skills:

Algorithmic thinking; teamwork.

Student Prerequisites:

Students should be able to (1) alphabetize, and (2) manipulate basic flowchart symbols (if the flowcharting option is chosen).

Teacher Prerequisites:

None.

Materials:

3" x 5" cards; box, such as a shoe box.

Time Required:

One or two class periods.

Source:

Reston and Hunter (1983)

Preparation:

(1) Choose five students, line them up, and give each a 3 x 5 card.

(2) The first child on line puts "E" on the card, the second child a "D", the third child a "C", the fourth a "B", and the fifth an "A".

The five students are now lined up in the following order: E, D, C, B, A.

(3) Object: To reverse the order of the cards (not the students) so that E, D, C, B, A becomes A, B, C, D, E, without changing the position of the students. Do not tell the students what the goal is. Their task is to figure out how to trace the algorithm.

Rules:

(1) A student may hold only one card at a time.

(2) If a student is given a second card, the first card is voided and may not be used until the next sort is attempted.

(3) The shoe box may hold only one card at a time.

Performing the sort:

(1) Put the card held by student number 1 into the shoe box.

(2) Give the card held by student number 5 to student number 1.

(3) Student number 5 takes the card from the shoe box.

(4) Student number 2 puts his/her card into the shoe box.

(5) Student number four gives his/her card to student number 2.

(6) Student number 4 takes the card from the shoe box.

Variations

(1) Use numbers or words instead of letters.

(2) Divide the class into teams of five, give each team some data to sort, and see which team creates the most efficient sort (i.e., the sort requiring the fewest number of transfers).

(3) Use the students' names on the cards.

(4) Have the students develop their own algorithms to alphabetize a series of names not in alphabetical order.

(5) Have the students perform the following flowchart activity.

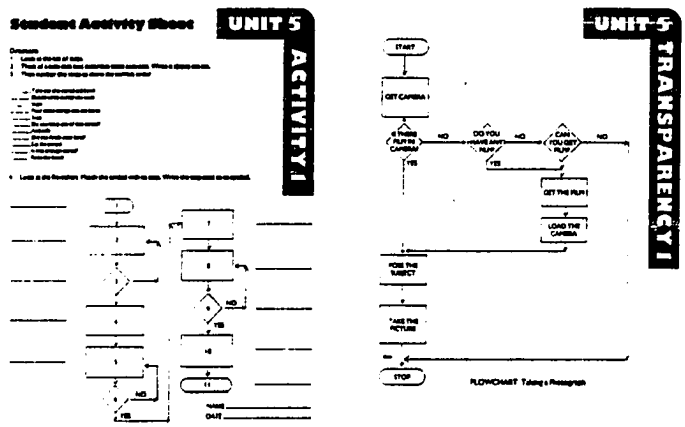
UNIT 5 LESSON 1

OBJECTIVE

To introduce flowcharting as a way to represent the sequential steps necessary to complete a task

PREPARATION AND MATERIALS

1. Student Activity Sheet 5—1
2. Transparency 5—1



LESSON

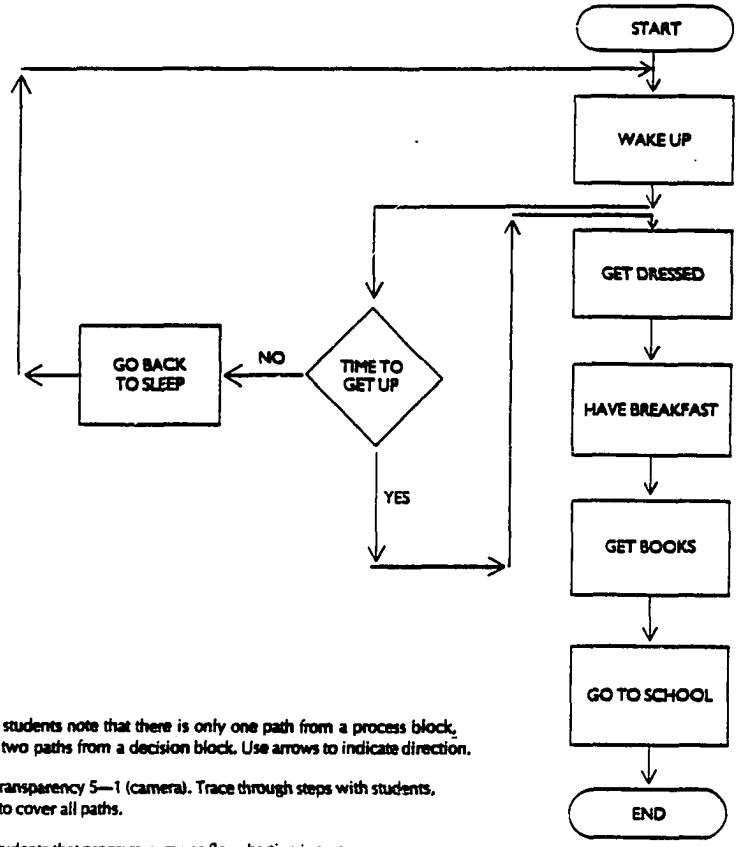
1. Ask the students to list their morning activities in five steps:
   - Wake up
   - Get dressed
   - Have breakfast
   - Get books
   - Go to school

2. Draw the flow chart on the chalkboard.

Use ⬭ to indicate start and end. Use ▭ to represent a process or activity.

3. Introduce the decision block ◇ by altering the previous flowchart as shown on following page.

Figure 2. Activity 5.



Have the students note that there is only one path from a process block, and only two paths from a decision block. Use arrows to indicate direction.

4. Display Transparency 5—1 (camera). Trace through steps with students, being certain to cover all paths.

5. Tell the students that programmers use flowcharting in two ways:
   - To show how a program works (documentation)
   - To plan a program

6. Distribute Student Activity Sheet 5—1. Direct the students to complete the assignment independently.

BACKGROUND INFORMATION

The best way to plan a program is to make a picture displaying its logic. The picture that programmers use is called a flowchart. Because flowcharts are so widely used, programmers have devised standard symbols. The most common ones are:

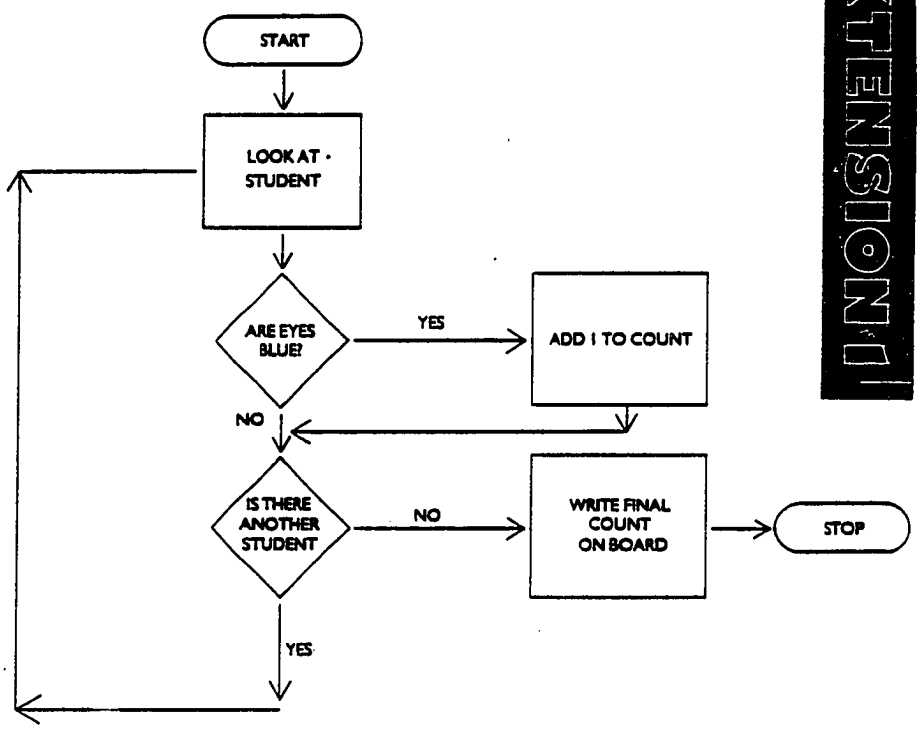
Flowcharts are most helpful in designing programs that are kept simple. A complex or cluttered flowchart is difficult to read. A flowchart is supposed to help the designer—not be more work than it's worth. It is helpful in planning the *logic* of a program.

Figure 2. Activity 5 (Continued).

UNIT 5 EXTENSION

EXTENSION ACTIVITIES

1. Ask students to think of the steps needed to find the number of blue-eyed students in the class. Direct students to flowchart the solution. Emphasize the concept of looping to include all the students in the count.



Assign students to write a flowchart that will show the steps necessary to find out how many students have summer birthdays.

2. Have students design flowcharts to show solutions to the following:

   - How to tie a bow
   - Putting on shoes and socks
   - Finding the average of three numbers
   - Selecting a library book

3. Ask students to list 5 activities that they enjoy. Then select one activity to flowchart.

Figure 2. Activity 5 (Continued).

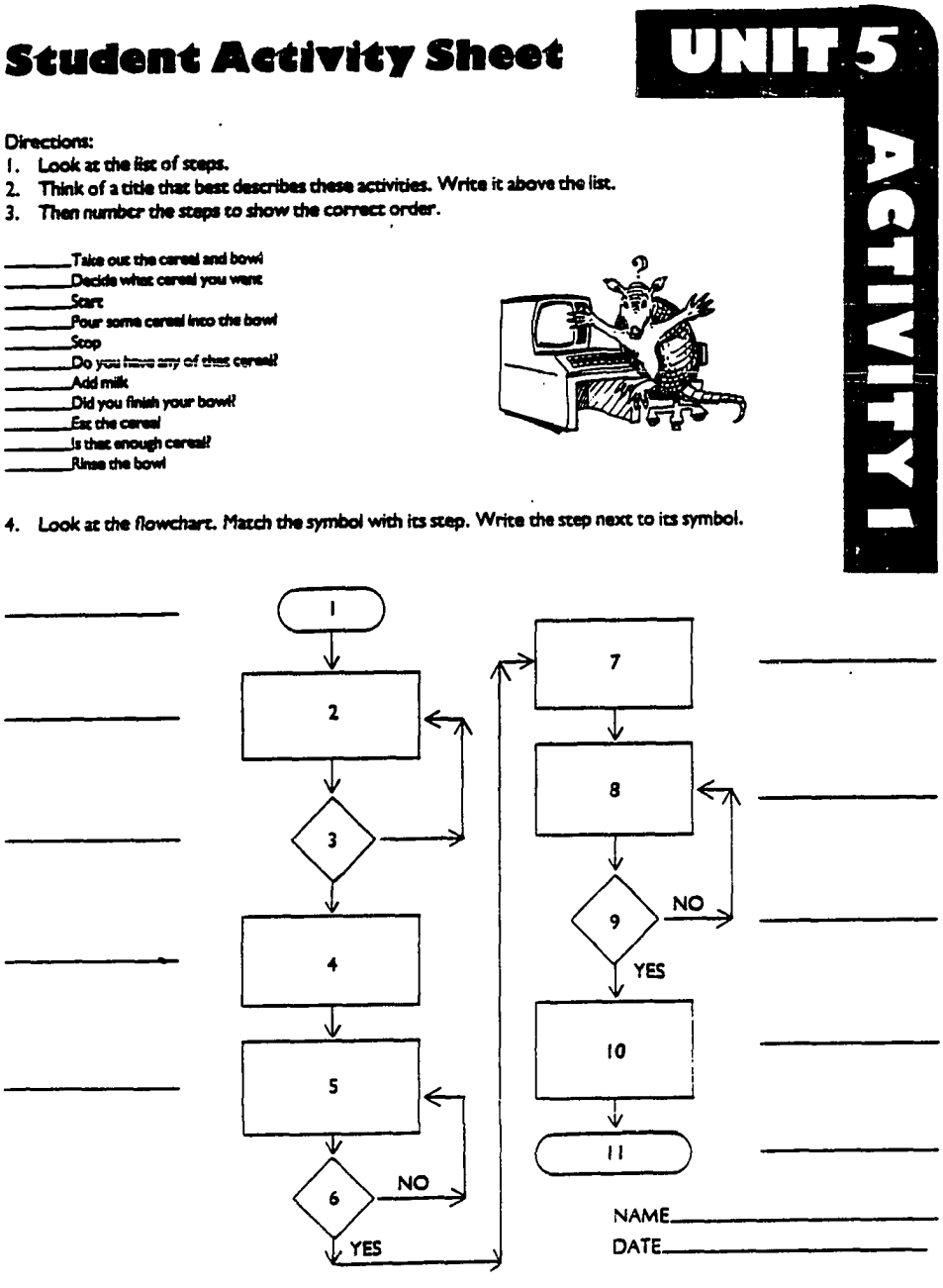


Figure 2. Activity 5 (Continued).

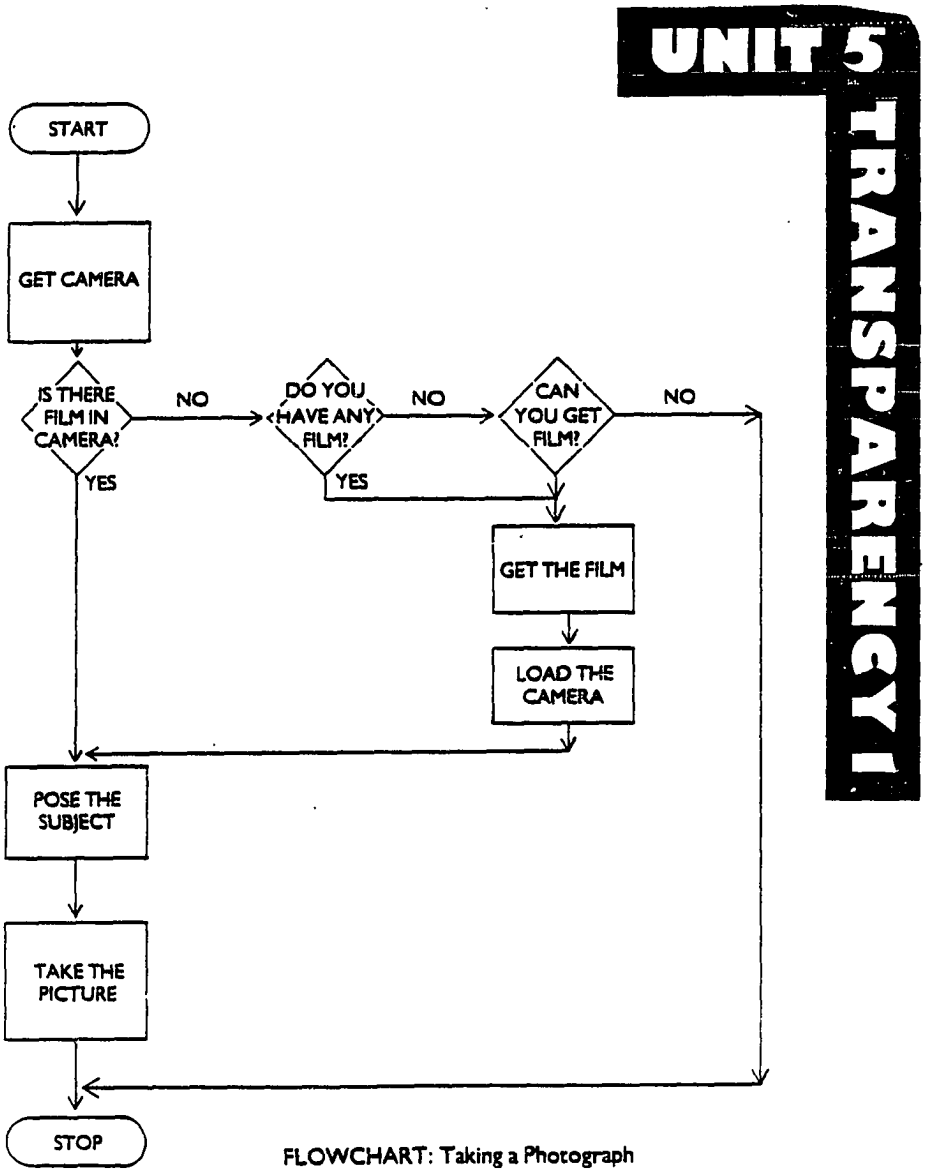


Figure 2. Activity 5 (Continued).

The author taught this activity to a group of six fourth grade Pace students at McCornack Elementary School in Eugene, Oregon. The idea for the activity was obtained from the galleys of a book by Reston and Hunter (1983). The activity was modified by adding Parts four and five. The flowcharting exercise was taken from Computers Demystified, an Elementary School Computer Literacy Book published by the Houston Independent School District.

This activity was selected because it gives children the opportunity to work in teams to trace a fairly difficult algorithm. The algorithm chosen seems far enough removed from the everyday experiences of most elementary school students that the "answer" will not be guessed immediately after performing the first step. Thus, the students will actually have to trace through the entire algorithm in order to obtain the answer. They will have to work closely with their teammates.

Activity: Debugging

Purpose: To give students practice in finding and debugging errors in algorithms and programs.

Specific Skills:

Debugging.

Student Prerequisites:

Experience with algorithmic thinking;

programming ability (if the programming option is chosen).

Teacher Prerequisites:

Programming ability (if the programming option is chosen).

Materials:

Microcomputer (for programming option); watercolors and other painting supplies.

Time Required:

One class period.

Source:

Papert (1980)

Directions: Find and correct the errors in the following directions for painting a picture with a box of watercolor paints:

(1) Open the box of paints.
(2) Fill a cup with water.
(3) Get a piece of paper to paint on.
(4) Close the box of paints.
(5) Wet the brush in the cup of water.
(6) Wet the color you want to paint with.
(7) Paint.
(8) Clean the brush in the cup of water.
(9) Repeat Steps 4-8 until the painting is finished.

Put these directions in the proper order:

(1) Set up the easel.

(2) Clean up.

(3) Dip the brushes in the paint.

(4) Assemble the paints and other supplies.

(5) Hang the painting to dry.

(6) Spread the paper on the floor.

(7) Paint the picture.

(8) Put the smock on.

Put these directions for washing dishes into the proper order:

(1) Put detergent into the water.

(2) Put the plug in the drain.

(3) Rinse the detergent off the cleaned dishes.

(4) Clean the dirty dishes with a sponge.

(5) Place the dirty dishes into the water.

(6) Fill the sink with water.

(7) Dry the cleaned dishes.

Programming Option

The following program is supposed to add the numbers from 1 to 10. Unfortunately, it does not produce the desired result. Find and correct the error(s).

```
10  FOR I = 1 TO 10
20  LET SUM = 0
30  LET SUM = SUM + 1
40  NEXT I
50  PRINT SUM
60  END
```

These activities were tested on a group of seven fifth graders at Westmoreland Elementary School in Eugene, Oregon. The idea for this activity was planted by Seymour Papert in Mindstorms. In this book, Papert discusses at length an important change he hopes computers will bring to education, namely, the move away from an emphasis on answers being right or wrong toward a concentration on finding and eliminating bugs.

The goal of this activity is to get students thinking about how to correct errors rather than having them dwell upon the correctness or error of a particular answer. Computers demand the ability to find and correct errors. This exercise is intended to give students practice in this important skill.

Activity: Logical Operators

Purpose: To give students practice using logical and and or.

Specific Skills:

Precision; algorithmic thinking.

Student Prerequisites:

None.

Teacher Prerequisites:

None.

Materials:

Dittoed copies of the problems.

Time Required:

One class period.

Source:

Author

Directions: Write the following statements on the board or distribute dittoed copies to each student. Then, discuss the questions that follow each statement.

(1) <u>If the class behaves well at the assembly, the class will have a party</u>.

What will result if:

(a) The class constantly talks during assembly?

(b) The class sits quietly and listens?

(c) The class does not sit quietly?

(d) The play is bad (not enough information is given)?

(2) <u>If the class behaves well at the assembly and returns to the classroom by 2:00 p.m., the class will have a party</u>.

What will result if:

(a) The class behaves well and returns at 1:45 p.m.?

(b) The class returns at 2:15 p.m. after behaving well?

(c) The class behaves poorly and returns at 1:00 p.m.?

(d) The class behaves poorly?

(e) The class behaves well, returns at 2:02 p.m.? (Technically, no party. But, what would probably happen?)

(3) <u>If the class behaves well or we return by 2:00 p.m., the class will have a party</u>.

What will result if:

(a) The class misbehaves and returns at 3:00 p.m.?

(b) The class misbehaves and returns at 1:00 p.m.?

(c) The class behaves well but does not return until 3:00 p.m.?

(d) The class behaves well and returns at noon?

General rule: If a compound statement consists of two statements joined by <u>and</u>, the compound statement is true only if both of the shorter component statements are true. If the two statements are joined by <u>or</u>, the compound

statement is true if one or both of the component statements is true.

Variations

(1) Have the students make up their own compound statements using and or or.

The author taught this exercise to two groups of students in Engene, Oregon: a group of five regular fifth graders at Westmoreland and a group of four fourth graders at McCornack Elementary School.

This activity was developed by the author with the specific intention of developing the precise type of thinking patterns that logical operators demand. Working with and or or requires a type of thinking that most students have experienced only informally in their daily lives. The aim here is to formalize these thought patterns.

As mentioned above, each of these activities was taught by the author to at least one, and usually two, groups of students, From the author's standpoint, the classes went well. The students seemed interested and attentive. Their responses on the questionnaire supported this impression. These responses are summarized in Step 4.

Step 4: Interview Students.

After each small group lesson, the participating students were interviewed. Each student was asked to complete a questionnaire about the activity he or she experienced (see Appendix). The results were as follows:

N: 52

DID YOU LIKE IT?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

1 - 67%
2 - 25%
3 - 4%
4 - 4%
5 - 0%

WOULD YOU LIKE TO DO IT AGAIN?

Yes - 80%
No - 20%

WAS IT?

Hard - 0%
Just Right - 67%
Easy - 33%

After collecting the questionnaires, the author led the group in a discussion of the activity just completed, focusing upon what the students thought they had learned from it and upon whether or not they had enjoyed the activity. By and large, the children were quite capable of expressing their likes and dislikes about an activity. They were free with their opinions, as the following representative answers reveal (in response to the question "What did you learn from participating in this activity?):

> That it is possible to program on anything, whether or not it's on paper or computer.
>
> Nothing.
>
> How to play Tic-Tac-Toe better than I used to.
>
> How to follow directions a little bit more.
>
> I think I learned something.
>
> You have to make directions more specific.
>
> A bug was in the program.
>
> I'm not sure.
>
> Nothing.
>
> Zilch.
>
> Nothing.
>
> A lot.
>
> There are two sides to a question.
>
> Everybody has different opinions.
>
> Programming.

That certain people never lose at Tic-Tac-Toe.

I learned what an algorithm is.

Debugging.

Some computer arithmetic.

Step 5: Interview Teachers to Obtain Their Opinions About the Activities.

Two groups of teachers were asked to read the activities and fill out a questionnaire. The first group consisted of 15 elementary school Teacher Technologists from the Houston Independent School District.

The Teacher Technologist program is a program designed to provide the Houston Independent School District with a cadre of trained personnel to teach the District's computer literacy curriculum. The goal is to place at least one Technologist in every one of the Houston Independent School District's 246 schools. Selection to the program is competitive from among H.I.S.D.'s classroom teachers, with those selected receiving 296 hours of inservice training at the District's Department of Technology training center (see Appendix for a more detailed description of the Technologist's role and their training sequence).

The Technologists interviewed were attending a week-long training session, four days of which were devoted to

LOGO and one day to the activities developed in STEP 3 of this dissertation. This workshop will be explained in greater detail in STEP 6.

The second group consisted of eight regular elementary school teachers from the Houston Independent School District. These teachers were participating in a three-day workshop in basic hands-on activities. H.I.S.D. has a requirement that before a school can receive a microcomputer, the principal and at least one teacher must have taken a basic hands-on workshop. The workshop the second group attended was for teachers only.

At the conclusion of these workships, each participant was asked to answer a questionnaire (see Appendix). The results were as follows:

AVERAGE NUMBERS OF YEARS TEACHING EXPERIENCE:

GROUP 1: 13.9 Years

Standard deviation:

Range: 2 - 29

GROUP 2: 14.5 Years

Standard deviation:

Range: 1 - 30

QUESTION #1: THESE ACTIVITIES WILL BE BENEFICIAL TO ELEMENTARY SCHOOL STUDENTS?

Explanation of Scale: 1 - Strongly agree

2 - Agree

Explanation of Scale: 3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 47%
2 - 53%
3 - 0%
4 - 0%
5 - 0%

NUMBER RESPONDING: 15
MEAN: 1.5333
STANDARD DEVIATION: .5164

GROUP 2 (REGULAR TEACHERS)

1 - 37.5%
2 - 37.5%
3 - 25%
4 - 0%
5 - 0%

NUMBER RESPONDING: 8
MEAN: 1.8750
STANDARD DEVIATION: .8062

QUESTION #2: ELEMENTARY SCHOOL STUDENTS WILL ENJOY THESE ACTIVITIES?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 27%
2 - 73%
3 - 0%
4 - 0%
5 - 0%

NUMBER RESPONDING: 15
MEAN: 1.7333
STANDARD DEVIATION: .4577

GROUP 2 (REGULAR TEACHERS)

1 - 37.5%
2 - 50%
3 - 13%
4 - 0%
5 - 0%

NUMBER RESPONDING: 8
MEAN: 1.7500
STANDARD DEVIATION: .6831

QUESTION #3: ELEMENTARY SCHOOL TEACHERS WILL BE ABLE TO USE THESE ACTIVITIES IN THEIR CLASSROOMS IF THEY RECEIVE SUFFICIENT TRAINING?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 33%
2 - 60%
3 - 0%
4 - 7%
5 - 0%

NUMBER RESPONDING: 15
MEAN: 1.8000
STANDARD DEVIATION: .7746

GROUP 2 (REGULAR TEACHERS)

1 - 37.5%
2 - 37.5%
3 - 25%
4 - 0%
5 - 0%

NUMBER RESPONDING: 8
MEAN: 1.8750
STANDARD DEVIATION: .7800

QUESTION #4: A ONE OR TWO-DAY WORKSHOP WILL BE SUFFICIENT TO TRAIN AN ELEMENTARY SCHOOL TEACHER TO USE THESE ACTIVITIES?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 36%
2 - 29%
3 - 14%
4 - 21%
5 - 0%

NUMBER RESPONDING: 14
MEAN: 2.2143
STANDARD DEVIATION: 1.1883

GROUP 2 (REGULAR TEACHERS)

1 - 25%
2 - 50%
3 - 12.5%
4 - 0%
5 - 12.5%

NUMBER RESPONDING: 8
MEAN: 2.2500
STANDARD DEVIATION: 1.2817

QUESTION #5: TEACHER TECHNOLOGISTS HAVE AN ADEQUATE BACKGROUND TO SHOW OTHER TEACHERS HOW TO USE THESE ACTIVITIES?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 36%
2 - 36%
3 - 21%
4 - 7%
5 - 0%

NUMBER RESPONDING: 14
MEAN: 2.0000
STANDARD DEVIATION: .9608

GROUP 2 (REGULAR TEACHERS)

1 - 50%
2 - 37.5%
3 - 12.5%
4 - 0%
5 - 0%

NUMBER RESPONDING: 8
MEAN: 1.6250
STANDARD DEVIATION: .7440

QUESTION #6: THESE ACTIVITIES ARE SUITABLE FOR INTELLECTUALLY GIFTED STUDENTS?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 20%
2 - 60%
3 - 6%
4 - 14%
5 - 0%

NUMBER RESPONDING: 15
MEAN: 2.1333
STANDARD DEVIATION: .9155

GROUP 2 (REGULAR TEACHERS)

1 - 50%
2 - 37.5%
3 - 0%
4 - 12.5%
5 - 0%

NUMBER RESPONDING: 8
MEAN: 1.7500
STANDARD DEVIATION: 1.0351

QUESTION #7: THESE ACTIVITIES ARE SUITABLE FOR AVERAGE STUDENTS?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 27%
2 - 73%
3 - 0%
4 - 0%
5 - 0%

NUMBER RESPONDING: 15
MEAN: 1.7333
STANDARD DEVIATION: .4577

GROUP 2 (REGULAR TEACHERS)

1 - 37.5%
2 - 12.5%
3 - 37.5%
4 - 12.5%
5 - 0%

NUMBER RESPONDING: 8
MEAN: 2.2500
STANDARD DEVIATION: 1.1650

QUESTION #8: THESE ACTIVITIES ARE SUITABLE FOR SLOW STUDENTS?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 27%
2 - 67%
3 - 0%
4 - 6%
5 - 0%

NUMBER RESPONDING: 15
MEAN: 1.8667
STANDARD DEVIATION: .7432

GROUP 2 (REGULAR TEACHERS)

1 - 25%
2 - 12.5%
3 - 12.5%
4 - 50%
5 - 0%

NUMBER RESPONDING: 15
MEAN: 2.8750
STANDARD DEVIATION: 1.3562

QUESTION #9: THESE ACTIVITIES ARE SUITABLE FOR K-3 STUDENTS?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 16.5%
2 - 25%
3 - 42%
4 - 16.5%
5 - 0%

NUMBER RESPONDING: 12
MEAN: 2.5833
STANDARD DEVIATION: .9962

GROUP 2 (REGULAR TEACHERS)

1 - 12.5%
2 - 25%
3 - 37.5%
4 - 12.5%
5 - 12.5%

NUMBER RESPONDING: 8
MEAN: 2.8750
STANDARD DEVIATION: 1.2564

QUESTION #10: THESE ACTIVITIES ARE SUITABLE FOR STUDENTS IN GRADES 4-6?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 33%
2 - 67%
3 - 0%
4 - 0%
5 - 0%

NUMBER RESPONDING: 15
MEAN: 1.6667
STANDARD DEVIATION: 4.880

GROUP 2 (REGULAR TEACHERS)

1 - 37.5%
2 - 37.5%
3 - 25%
4 - 0%
5 - 0%

NUMBER RESPONDING: 8
MEAN: 1.8750
STANDARD DEVIATION: .8345

QUESTION #11: THESE ACTIVITIES ARE SUITABLE FOR STUDENTS IN GRADES 7-8?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 29%
2 - 57%
3 - 7%
4 - 7%
5 - 0%

NUMBER RESPONDING: 14
MEAN: 1.9286
STANDARD DEVIATION: .8287

GROUP 2 (REGULAR TEACHERS)

1 - 25%
2 - 25%
3 - 37.5%
4 - 12.5%
5 - 0%

NUMBER RESPONDING: 8
MEAN: 2.3750
STANDARD DEVIATION: 1.0607

QUESTION #12: THESE ACTIVITIES ARE SUITABLE FOR STUDENTS IN GRADES 9-12?

Explanation of Scale: 1 - Strongly agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly disagree

GROUP 1 (TEACHER TECHNOLOGISTS)

1 - 14%
2 - 29%
3 - 36%
4 - 7%
5 - 14%

NUMBER RESPONDING: 14
MEAN: 2.7857
STANDARD DEVIATION: 1.2514

GROUP 2 (REGULAR TEACHERS)

1 - 25%
2 - 25%
3 - 25%
4 - 12.5%
5 - 12.5%

NUMBER RESPONDING: 14
MEAN: 2.6250
STANDARD DEVIATION: 1.4079

When interpreting this data, it is helpful to set an arbitrary cutoff point at 2.0, with below this point being "Agree" and above this point being "Disagree." Using this cutoff point, the two groups agreed on questions 1-4, 9, 10, and 12, and disagreed on questions 5-8 and 11.

Further statistical analysis of this data is hindered by the small sizes of the samples and the differences between the two groups. The teachers who qualified for the Teacher Technologist program needed: three years teaching experience with above average evaluations, recommendations from their principals, and high scores on a rigorous interview. The regular teachers needed only to be teachers in a school located within the Houston Independent School District.

Step 6: Develop and Pilot Test a Workshop to Train a Selected Group of Teachers to Use the Activities Developed and Refined Previously.

Before a new curriculum can be taught, the teachers who are expected to teach it must be adequately trained. Two eight-hour workshops were taught at the Houston Independent School District's Department of Technology to accomplish this task. The participants consisted of the two groups of

teachers interviewed in STEP 5. The goals of these workshops were to:

(1) explain to the participants the steps that had gone into developing the activities;

(2) explain the educational purposes of the activities;

(3) discuss methods to teach the activities; and,

(4) critique the activities and exchange criticisms with other workshop participants.

The workshop began with a 45-minute lecture which outlined the development of the activities. Several of the activities were then explained in detail. The participants were then asked to spend an hour reading the activities. During the afternoon session, the participants broke into small groups to develop a detailed critique of one of the activities. Finally, the small groups presented their critiques to the entire class.

Those participants who knew BASIC were given the opportunity to experiment with the programming options.

At the conclustion of the workshop, each participant was asked to answer a questionnaire about the workshop (see Appendix). The results were as follows:

QUESTIONS:

(1) I chose to attend this workshop.

(2) The workshop met my needs.

(3) The activities were appropriate to communicate the content.

(4) The materials were effective to communicate the content.

(5) My knowledge of the topic was increased.

(6) The workshop would be valuable for someone in the area.

Explanation of Scale: 1 - Strongly disagree
2 - Disagree
3 - Don't know
4 - Agree
5 - Strongly agree

GROUP 1 (TEACHER-TECHNOLOGISTS)

NUMBER RESPONDING: 11

QUESTION	MEAN
#1	4.64
#2	4.00
#3	4.09
#4	4.00
#5	4.45
#6	4.18

GROUP 2 (REGULAR TEACHERS)

NUMBER RESPONDING: 8

QUESTION	MEAN
#1	4.88
#2	4.88

QUESTION	MEAN
#3	4.88
#4	4.75
#5	5.00
#6	4.75

QUESTIONS:

(7) The presenter was organized.

(8) The presenter was knowledgeable about the subject matter.

(9) The presenter had good communication skills.

Explanation of Scale: 1 - Poor
2 - Below average
3 - Average
4 - Above average
5 - Excellent

GROUP 1 (TEACHER TECHNOLOGISTS)

NUMBER RESPONDING: 11

QUESTION	MEAN
#7	4.09
#8	4.64
#9	4.09

GROUP 2 (REGULAR TEACHERS)

NUMBER RESPONDING: 8

QUESTION	MEAN
#7	4.63
#8	5.00
#9	4.75

Each of the teachers trained was given copies of the activities and asked to integrate them into their classrooms. Preliminary reports have been favorable.

CHAPTER FIVE

DISCUSSION

Why analyze the problem-solving skills used by computer scientists? One answer is that computers are becoming so important in many areas of our society that it is to our advantage to examine the types of skills that will be required to use computers most efficiently. Computer science is not like physics or chemistry in that most individuals do not have physics or chemistry labs in their homes. But, it appears that computers will soon be in most homes, much as telephones are today. Virtually all members of society will be impacted by the computer. People will work with computers, bank with them, and use them for recreation.

One belief upon which this dissertation is based is that the better people understand computers, the more efficiently and productively they will use them. To attain this goal, a method for developing a curriculum has been developed in this dissertation that stresses thought and consultation before action and design.

(1) STEP 1: Role of Computer Scientists in Pre-college Computer Science Curriculum Development.

The problem-solving methods used by computer scientists

can be defined by interviewing computer scientists. The computer scientists who agreed to be interviewed for this dissertation were quite capable of defining the problem-solving skills that they used in their work. The questions made sense to them. They all felt that computer science emphasized certain problem-solving skills more than others.

However, because computer science is such a technical field, it would be difficult for someone with little or no background in computer science to conduct a meaningful interview on this topic. To conduct such interviews requires an interviewer with a background in both computer science and education. The computer science background enables the interviewer to define questions that professional computer scientists can relate to. An education background is needed to help the interviewer focus the discussion on the educational issues pertaining to the topic.

As computer technology develops, the problem-solving skills used by computer scientists may shift in emphasis. For example, as database and telecommunications technologies become more widespread, the ability to sift through large quantities of data will undoubtedly grow in importance.

The developers of elementary school computer science curricula would do well to keep abreast of these changes. Educators should remember that computer science education is only part education. There is a computer science component

as well. To ensure that educators keep their knowledge up to date, the lines of communication between educator and computer scientist must be kept open.

In summary, to bridge the gap between education and computer science requires a group of professionals trained in both disciplines. If one believes that computer science has important contributions to make to problem solving, the case is made for developing a group of professionals capable of communication with both educators and computer scientists.

(2) STEPS 2 AND 3: Activities Suitable for Elementary School Students Can be Found or Developed.

There are many sources of activities for an elementary school computer science curriculum. However, much of what has been developed is of poor quality, and it takes considerable wading through this material by a knowledgeable wader to separate the good from the bad.

Why has so much poor quality material been produced? An answer seems to be that much of this material was developed by educators with little or no background in computer science (see Appendix E). Much of the material focuses on trivialities or over simplifications, such as binary arithmetic or the history of computing, as the developers concentrate upon what they understand rather than upon what professional computer scientists think

important. Other material stresses topics, such as BASIC programming, far too difficult for many--if not most--elementary school students. How can this problem be overcome? A helpful first step would be to follow the recommendation made in (1): to keep the lines of communication open between computer scientist and curriculum developer. The former are needed to ensure that the essentials are covered in the curriculum, while the latter help make the curriculum teachable, learnable, and enjoyable for the majority of elementary school students.

Despite the shortcomings of much of the material currently available, many high quality activities suitable for elementary students were found. Sources which yielded usable activities included books, journals, magazines, classroom teachers, and fellow students in computer science education.

But, to adequately cover each of the defined problem-solving skills, the author had to develop 6 of the 12 activities himself. The fact that 50 percent of the activities had to be developed by the author attests to the fact that many of the activities already developed were not based upon the view of problem solving developed here which, in turn, is based upon the problem-solving skills computer scientists defined as most important in their discipline. The author has had considerable training in both computer science and education, as well as extensive experience as a

classroom teacher. He was able to develop activities when one could not be found. Whether less well-trained teachers will be able to duplicate this effort remains to be seen.

(3) STEP 4: Student Feedback.

The fourth and fifth grade students who were interviewed for this dissertation were eager and willing to give their opinions when asked for them. These opinions were quite helpful in discovering whether or not an activity was enjoyable. However, when asked about what a particular activity was trying to get across, their opinions were virtually worthless. Children of this age do not seem to understand that education has profound intellectual goals; to them, the activities were either fun or not fun. They were incapable of articulating opinions which went any deeper than this, even though many of the students were from classes for the intellectually-gifted.

(4) STEP 5: Teacher Feedback Can Help.

Helpful feedback can be obtained from teachers regarding the teachability of the activities. A difference was noticed between the feedback given by teachers well trained in computer science education and those teachers relatively untrained.

Better trained teachers, such as the Houston Independent School District's Teacher Technologists, seemed to have different views regarding the activities than did their less

well-trained colleagues. There is some statistical evidence to support this conclusion, but no definite conclusions can be drawn from the data due to the large differences between between the two groups and the small sizes of the samples. The Technologists felt that the activities were usable with children of average or below agerage abilities but were not suitable for intellectually-gifted students. Untrained teachers, on the other hand, felt that the activities could be used with gifted children but not with below average or average children. In addition, the Technologists felt the activities were appropriate for students in grades four through six, while the untrained teachers felt the activities suitable for students in higher grades.

These findings indicate that as a teacher learns more about computers and computer science education, the activities in a computer science curriculum seem easier and, therefore, more appropriate for younger, less able students. Untrained teachers seem to fear the unknown and feel it is quite difficult to grasp.

(5) STEP 6: Developing a Workshop.

A one-day, eight-hour workshop can be developed to train teachers to use these activities. The teachers who participated in the pilot workshop seemed to enjoy the training and find it worthwhile.

Each of the workshop participants was given a copy of the activities and asked to spend the 1983-84 school year trying to integrate them into their teaching. Preliminary reports have been favorable but, until the end of 1983-84 school year, no hard data can be collected. At that time, a questionnaire will be distributed to the teachers who have agreed to try out the activities in their classrooms. See the Appendix for a model of what this questionnaire will resemble (see Appendix F).

Implications for Education

To help educators incorporate computer science problem-solving skills into the elementary school curriculum, the following steps should prove useful:

(1) Keep the lines of communication open between educators and computer scientists. Both groups have skills essential to incorporating computer science problem-solving skills into the elementary school curriculum. As discussed previously, a group of dedicated computer science educators is needed to bridge the gap.

(2) Train teachers of computer science education to the highest level possible in computer science. Teachers who know more about computers seem to be disposed to teach to

higher levels than their less-informed colleagues. The teachers in the Teacher Technologist program in the Houston Independent School District tended to think the activities were suitable for lower ability students than did teachers who had less training about computers. This training can take the form of college coursework at institutions, such as the University of Oregon, that have made a serious commitment to training teachers in this area, or inservice training, such as that provided by the Houston Independent School District's Department of Technology.

(3) Listen to teachers and students to find out what will work in the classroom. Students know what they like, and teachers have ideas about what will work in the classroom.

(4) Listen to computer scientists to discover the key problem-solving skills used in their work. They alone possess the depth of knowledge and the experience to provide this information.

Suggestions for Further Research

Further research is needed in the following areas:

(1) Do students learn the problem-solving

techniques that the activities are intended to teach them? This is a critical question in any area of education, but it is especially crucial when a new discipline is being introduced into the curriculum. When the children are taught to use these problem-solving techniques, do they actually use them or do they revert to other, more familiar problem-solving techniques? Protocol research, with one researcher working with a small number of students at a time, should prove useful in providing answers to these questions.

(2) Is there transfer of knowledge from the classroom to environments outside the classroom? Again, this is a crucial question for all educators. Even if the students learn the computer science problem-solving skills recommended in this dissertation, will they become better problem solvers in the non-academic world? Will they be better able to function in a computerized society? Until a valid measure of problem solving is developed, protocol research will probably be needed to begin answering these questions.

(3) Does training in computer science education change teacher attitudes toward computers?

Does it help them incorporate computer science into the curriculum that they follow in their classrooms? Do teachers who receive more training believe they can teach activities to younger, less able students?

Final Remarks

This dissertation has touched upon many topics. Curriculum development, problem solving, data collecting, and workshop development have all been covered. Any one of these topics could have been the basis for a separate dissertation. In such a situation, it is natural that some topics were covered less than comprehensively. It is difficult to deal with a topic as broad as elementary school computer science curriculum without omitting some topics that others feel should have been included.

Chief among these is assessment of student achievement. No attention was paid in this dissertation to assessment of student achievement, not because the author did not feel assessment important, but because it was simply too difficult a task to design a test to measure student achievement in the problem-solving skills essential to computer science. Problem solving is difficult to define, much less assess, and the thrust of this dissertation was to define the problem-solving methods themselves rather than methods on how to assess whether or not they have been learned. An

implementation of the model presented here in a school district would naturally have to include some assessment of student achievement. This is such a commonly held opinion in current educational thought that I did not think it necessary to include it in the model.

APPENDIX A

FACULTY INTERVIEW FORM

PROFESSOR:_____

Date:_____

How important are the following to a computer scientist?

(1) Intelligence

(2) Persistence

(3) Math Skills

(4) Curiosity

(5) Debugging Skills

(6) Top-Down-Design

(7) Programming Ability

(8) Precision

(9) Sequence

(10) Algorithmic Thinking

(11) Ability to Work in Teams

(12) Concerns for Social Impact of Computers

APPENDIX B

STUDENT ACTIVITY QUESTIONNAIRE

NAME:_____ SCHOOL:_____

DATE:_____

ACTIVITY:_____

(1) Did you like it?

(2) Was there anything about it you did not like? If so, what?

(3) Was it too hard? Too easy?

(4) Would you like to do this activity again?

(5) What did you learn?

APPENDIX C

TEACHER QUESTIONNAIRE

QUESTIONNAIRE ON ACTIVITIES

NAME:_____

DATE:_____

YEARS TEACHING EXPERIENCE:_____

Other than your Teacher Technologist training, do you have any other formal training in computer science? (Yes/No)
If yes, please describe.

Please use the following scale when answering the following questions.

1 - Strongly Agree
2 - Agree
3 - Neutral
4 - Disagree
5 - Strongly Disagree

1. ( ) These activities will be beneficial to elementary school students.

2. ( ) Elementary school students will enjoy these activities.

3. ( ) Elementary school teachers will be able to use

these activities in their classrooms if they receive sufficient training.

4. ( ) A one or two-day workshop will be sufficient to train an elementary school teacher to use these activities.

5. ( ) Teacher Technologists have an adequate background to show other teachers how to use these activities.

6. ( ) These activities are suitable for intellectually-gifted students.

7. ( ) These activities are suitable for average students.

8. ( ) These activities are suitable for slow students.

9. ( ) These activities are suitable for K-3 students.

10. ( ) These activities are suitable for students in grades 4-6.

11. ( ) These activities are suitable for students in grades 7-8.

12. ( ) These activities are suitable for students in grades 9-12.

APPENDIX D

WORKSHOP EVALUATION FORM

HOUSTON INDEPENDENT SCHOOL DISTRICT
DEPARTMENT OF TECHNOLOGY

WORKSHOP EVALUATION

Section 1-Circle one choice per statement.	Strongly Disagree	Disagree	Don't Know	Agree	Strongly Agree
. I chose to attend this workshop.	1	2	3	4	5
. The workshop met my needs.	1	2	3	4	5
. The activities were appropriate to communicate the content.	1	2	3	4	5
. The materials were effective to communicate the content.	1	2	3	4	5
. My knowledge of the topic was increased.	1	2	3	4	5
. The workshop would be valuable for someone in the area.	1	2	3	4	5

Section 2-Please list each presenter's name and circle the appropriate descriptors.

	ORGANIZATION					TOPIC KNOWLEDGE					COMMUNICATION SKILLS				
	Poor	Below Average	Average	Above Average	Excellent	Poor	Below Average	Average	Above Average	Excellent	Poor	Below Average	Average	Above Average	Excellent
PRESENTER															
#1_____	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
#2_____	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
#3_____	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5

SECTION 1

Last

6

First

22

SOCIAL SECURITY NUMBER:

34

SIGNATURE

SECTION 2

SCHOOL OR DEPARTMENT:

CAMPUS:

43

49

SECTION 3

PRIMARY JOB TITLE (Check One):

52

0 ☐ Teacher
1 ☐ Aide
2 ☐ Curriculum Coordinator/Supervisor
3 ☐ Principal/Assistant Principal
4 ☐ Central Office Administration
5 ☐ Guidance/Counselor
6 ☐ Parent/Volunteer
7 ☐ Food Services
8 ☐ Health Services
9 ☐ Other-Please specify:

10 ☐ Area Officer

PRIMARY SPECIALIZATION AREA (Check One):

53-54

01 ☐ General (all areas)
02 ☐ Title I-Regular
03 ☐ Title I-Migrant
04 ☐ Science
05 ☐ Social Studies
06 ☐ Reading/Language Arts
07 ☐ Mathematics
08 ☐ Physical Education
09 ☐ Bilingual
10 ☐ Migrant
11 ☐ Special Education
12 ☐ Vocational
13 ☐ Fine Arts
14 ☐ Driver Education
15 ☐ Other-Please specify:

PRIMARY ASSIGNMENT LEVEL (Check One):

55

0 ☐ Early Childhood
1 ☐ Elementary
2 ☐ Elementary-Primary
3 ☐ Elementary-Intermediate
4 ☐ Secondary
5 ☐ Secondary-Junior Middle
6 ☐ Secondary-High School
7 ☐ All levels or combination

APPENDIX E

EXAMPLE OF A MODULE DEVELOPED BY A CURRICULUM DESIGN SPECIALIST WITHOUT COMPUTER SCIENCE BACKGROUND

HIGH LEVEL AND LOW LEVEL LANGUAGES

MODULE ___-___-___

PREVIEW

Computers are silent pieces of equipment unless given instructions. How can you talk to a computer to tell it what to do? There are a number of ways to do just that. Once you learn to give the computer instructions, then you gain command of the powers of this marvelous machine.

OBJECTIVES

Upon completion of this module, you should be able to:

(1) Define a computer language.

(2) Distinguish between high level and low level languages.

PRE-REQUISITES

Module ___-___-___

MATERIALS

None

DIRECTIONS

Self-directed reading activity.

RESOURCES/OPTIONAL ACTIVITIES

Learning Packet ___-___-___

An Introduction to Microcomputers, Adam Osborne, David Bunnell, Osborne/McGraw Hill, Berkeley, California (1982).

The Beginner's Guide to Computers, Robin Bradbeer, Peter DeBono, Peter Laurie, Addison-Wesley Publishing Company, Reading, Massachusetts (1982).

EVALUATION

Post Test

WHAT IS A LANGUAGE?

Bonjour Hello Buenos dios

...are all ways of saying the same thing. They are all ways of communicating the same thought or idea but in different languages.

Each of these languages uses its own set of words. These words are called a vocabulary. Each language combines vocabulary words to express an idea or communicate.

Each language also has its own set of rules. This set of rules is called a syntax. We follow that syntax when we combine vocabulary words in order to communicate an idea.

> These words "I will go to the store" have meaning.
>
> These words "To the store I will go" have meaning.
>
> However, the words "the store to I go will" have no meaning.

The vocabulary is the same for all three groups of words. However, the rules of syntax were not followed in combining the words, "the store to I go will."

****

Complete the following descriptions of a language.

o A language is a method of _____.

o Each language has a set of words, called a _____.

o Each language has a set of rules called _____, that governs how vocabulary is combined.

() communicating, vocabulary, syntax

****

WHAT IS A COMPUTER LANGUAGE?

Actually you know what a computer language is because you understand what a "people" language is. A "people" language is a way for people to communicate with other people.

****

Complete the following description of a computer language.

o A computer language is a way for people to _____ with a computer.

o Just as a "people" language uses a vocabulary and follows the rules of syntax, a computer language uses a _____ and follows the rules of _____.

() communicate, vocabulary, syntax

****

Just as there are many languages people use to communicate:

German, French, Spanish, etc. etc. etc....

there are many languages to communicate with a computer:

BASIC, Pascal, COBOL, etc. etc. etc....

Programmers combine vocabulary using the rules of syntax to give a set of instructions to the computer. This set of instructions is called a program.

****

Complete the following description of communicating with a computer.

- o A program is a set of _____ given to the _____.
- o The people who write computer programs are called _____.

() instructions, computer, programmers

****

This program or set of instructions is input into the computer.

But....

the computer can only use a program that is in machine language.

WHAT IS A MACHINE LANGUAGE?

A computer is a machine, just as the human body is a machine. When we input chocolate ice cream into the human machine (some people call that eating), the human machine changes the proteins and carbohydrates of the ice cream into energy. The cells use this energy for food.

When we input a program into the computer machine, the computer machine translates everything into number codes made of 1's and 0's. This series of 1's and 0's is called Machine Language. Once this translation into 1's and 0's is done, the computer is able to use this machine language to carry out the instructions given to it by the programmer.

CHANGING A BASIC PRINT STATEMENT INTO MACHINE LANGUAGE

If we input a BASIC statement like:

PRINT "HELLO" and press return,

the computer translates it into Machine Language like this:

1001000	(H)
1000101	(E)
1001100	(L)
1001100	(L)
0110000	(O)

and HELLO would be displayed.

****

Complete the following statements.

o A computer can work only in _____.

o Machine language is a series of _____'s and _____'s.

() Machine Language, 0's, 1's

****

Here is an example of a Machine Language program. Can you determine what it tells the computer to do?

illustrate puzzlement

Most students will not be able to determine what the program tells the computer to do. This example illustrates the major drawback of Machine Language programming:

> It is very hard for humans to use just 1's and 0's to communicate.

If you were able to figure it out, then:

____ You already know Machine Language programming.

____ You are a genius.

____ Or, you have read ahead.

ASSEMBLY LANGUAGE

Here is that same program. This time, it is written in a language called Assembly Language.

```
CLE
ADD DATA 1
ADD B
PIM ANS
STP
DATA1:3
B:4
ANS:0
```

List any English-type abbreviations in the above program.

_____, _____, _____

Can you determine what this program written in Assembly Language tells the computer to do? You may check your answer a little further on.

Here is the same program written in BASIC language.

```
10 LET A = 3
30 LET B = 4
50 LET C = A + B
60 PRINT C
```

****

EXERCISE

o List the English language words you can find in the program written in BASIC.

_____, _____

o Can you determine what the program tells the computer to do? _____

() let, print

() Add two numbers and print the results.

****

****

Count the number of lines in each of these programs.

Machine Language program _____

Assembly Language program _____

BASIC Language program _____

We will refer to these numbers later on.

****

APPENDIX F

ACTIVITY EVALUATION QUESTIONNAIRE

NAME: _____

DATE: _____

GRADE TAUGHT: _____

SCALE

1 - Very good

2 - Good

3 - Neutral

4 - Bad

5 - Very bad

ACTIVITY	EASE OF TEACHING	RESPONSE OF STUDENTS	ADEQUACY OF TRAINING
Pred. Out.			
Soc. Con.			
Directions			
Spiffy			
Inter. Alg.			
T-T-T			
Alg. Dev.			
Hierarchies			
STROBS			
Tracing			
Debugging			
Log. Oper.			

REFERENCE LIST

Bantock, G. H., Dilemmas of the Curriculum, John Wiley and Sons, New York, New York (1980).

Bitter, Gary, "The Road to Computer Literacy: A Scope and Sequence Model," Electronic Learning (September, 1982, February, 1983).

Cupertino Union School District, "K-8 Computer Literacy Curriculum (revised 1982), The Computing Teacher (March, 1983) 7-10.

"Curriculum '78: Recommendations for the Undergraduate Program in Computer Science," Communications of the ACM (March, 1979) Vol. 22, No. 3.

Department of Technology, Houston Independent School District, Technology Planning Document, Houston, Texas (1982).

De Rossi, Claude J., Computers: Tools for Today, Children's Press, Chicago, Illinois (1972).

Fisher, Glenn, "Developing a District-Wide Computer Use Plan," The Computing Teacher (January, 1983) 52-59.

Frates, Jeffrey and Noldrup, William, Computers and Life, Prentice-Hall, Englewood Cliffs, New Jersey (1983).

Goodlad, John I. and Klein, M. Frances and Associates, Behind the Classroom Door, Charles A. Jones Publishing Company, Worthington, Ohio (1970).

Goodlad, John I. and Associates, Curriculum Inquiry: The Study of Curriculum Practice, McGraw-Hill Book Company, New York, New York (1979).

Graham, Neil, The Mind Tool, West Publishing Company, St. Paul, Minnesota (1983).

Hayes, John R., in Tuma, D. T. and Reif, Editors, Problem Solving and Education: Issues in Teaching and Research, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey (1980).

Houghton Mifflin Mathematics, Houghton Mifflin Company, Palo Alto, California (1981).

Hunter, Beverly, An Approach to Integrating Computer Literacy Into the K-8 Curriculum, Human Resources Research Organization, Alexandria, Virginia, National Science Foundation, Washington, D.C. (October, 1980, Ed. 195 247).

ICCE (International Council for Computers in Education), see Rogers.

Larkin, Jill H., in Tuma, D. T. and Reif, Editors, Problem Solving and Education: Issues in Teaching and Research, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey (1980).

Keating, John et. al., Introductory In-Service Workshop: Computers in Education, Department of Computer and Information Science, Eugene, Oregon (1982).

King, Arthur R., Jr. and Brownell, John A., The Curriculum and the Disciplines of Knowledge, Wiley, New York, New York (1966).

Markle, Sandra, and Armstrong, Bev, Computer Tutor: The Learning Works, Inc. (1981).

Moursund, David, Basic Programming for Computer Literacy, McGraw-Hill.

Moursund, David, Introduction to Computers in Education for Elementary and Middle School Teachers, ICCE, Eugene, Oregon (1981).

Moursund, David, Precollege Computer Literacy: A Personal Computing Approach, ICCE, Eugene, Oregon (1981).

Moursund, David, Teacher's Guide to Computers in Elementary School, ICCE, Eugene, Oregon (1980).

National Commission on Excellence in Education, A Nation at Risk, U. S. Department of Education, Washington, D.C. (April, 1983).

Newsweek, (May 9, 1983).

New York Times, "Employment Outlook in High Technology," (March 27, 1983) 61.

Pratt, David, Curriculum: Design and Development, Harcourt Brace Jovanovich, Inc., New York, New York (1980).

Prism: Priorities in School Math: Executive Summary of the Prism Project, NCTM, Inc., Reston, Virginia (1981).

Reif, Frederick, in Tuma, D. T. and Reif, Editors, Problem Solving and Education: Issues in Teaching and Research, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey (1980).

Robinson, Mary, Career Education Math: Units for Career Exploration in Sixth, Seventh, or Eighth Grade, Oklahoma State Department of Vocational and Technical Education, Stillwater, Oklahoma (1974).

Rogers, Jean B., An Introduction to Computers and Computing, International Council for Computers in Education, University of Oregon, Eugene, Oregon (1981).

Rubinstein, Moshe F., in Tuma, D. T. and Reif, Editors, Problem Solving and Education: Issues in Teaching and Research, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey (1980).

Schwab, Joseph J., "The Practical 3: Translating into Curriculum," School Review (August, 1973) 501-522.

Shuster, Albert H. and Ploghoft, Milton E., The Emerging Elementary Curriculum, Third Edition, Charles E. Merrill Publishing Company, Columbus, Ohio (1977).

Stern, Nancy and Stern, Robert A., Computers in Society, Prentice-Hall, Englewood Cliffs, New Jersey (1983).

Sturdivant, Patricia, Associate Superintendent, Houston Independent School District, interviewed in Portland, Oregon (May 12, 1983).

Taba, Hilda, Curriculum Development: Theory and Practice, Harcourt, Brace & World, Inc., New York, New York (1962).

Taylor, Robert P., Editor, The Computer in the School: Tutor, Tool, Tutee, Teachers College Press, New York, New York (1980).

The Tennessean, (March 13, 1983).

Tuma, D. T. and Reif, Editors, Problem Solving and Education: Issues in Teaching and Research, Lawrence Erlbaum and Associates, Publishers, Hillsdale, New Jersey (1980).

Tyler, Ralph W., Basic Principles of Curriculum and Instruction, University of Chicago Press, Chicago, Illinois (1949).

Weyer, S. A., and Cannara, A. B., Children Learning Computer Programming: Experiments with Language Curricula and Programmable Devices, Stanford University Institute for Mathematical Studies in Social Science, Palo Alto, California (1974, Ed. 111 347).

Wiles, Jon and Bondi, Joseph Jr., Curriculum Practice: A Guide to Practice, Charles E. Merrill Publishing Company, Columbus, Ohio (1979).

Willerding, Margaret F., From Fingers to Computers, Lyons & Carnahan, Chicago, Illinois (1970).

WLOCNC (West Linn, Lake Oswego, Oregon City, North Clackamas School District), The Do I Have to Teach Computer Literacy Handbook (1982).