ED 249 928 IR 011 352

AUTHOR Kurland, D. Midian
TITLE Educational Software Tools: Designing a Text Editor for Children. Technical Report No. 8.
INSTITUTION Bank Street Coll. of Education, New York, NY. Center for Children and Technology.
PUB DATE Apr 83
NOTE 12p.; Paper presented at the Annual Meeting of the American Educational Research Association (Montreal, Canada, April 11-15, 1983). For related document, see IR 011 340.
PUB TYPE Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Computer Software; *Design Requirements; Educational Research; *Elementary Education; *Instructional Design; Instructional Materials; *Material Development; *Word Processing

ABSTRACT

An overview of the research and design process that led to the creation of the Bank Street word processing program for children is reported. The decision to create a word processor for the classroom was based on the findings that commercially available programs took too long to learn, required the memorization of too many commands or rules, had difficult-to-read screen displays, and made editing so clumsy that the whole point of using the word processor to revise text was lost. The design team established a set of design principles: no control commands would be used; there would be comprehensive screen prompts; the display would allow upper and lower case letters; and no hardware modifications or additions would be needed in order for the program to run on a typical school computer system. Functions that the program would and would not need to carry out were identified. The design of the program serves as a starting point for a general discussion of desirable features and design characteristics for good educational software. Specific points covered include how to design educational tools and the features these tools and their manuals should have. (Author/THC)

CENTER FOR CHILDREN AND TECHNOLOGY

Educational Software Tools: Designing a Text Editor for Children

D. Midian Kurland

Technical Report No. 8

Bank Street College of Education 610 West 112th Street NY, NY 10025

Educational Software Tools: Designing a Text Editor for Children

D. Midian Kurland

Technical Report No. 8

ER011352

CENTER FOR CHILDREN AND TECHNOLOGY
Bank Street College of Education
610 West 112th Street
New York, NY 10025

EDUCATIONAL SOFTWARE TOOLS: DESIGNING A TEXT EDITOR FOR CHILDREN*

D. Midian Kurland

In a recent survey, Electronic Learning magazine (October 1982) asked 2,000 teachers and administrators to list their favorite software programs. The programs offered by those who responded were first subdivided according to whether they were intended for classroom or professional (i.e., administrative) use. The classroom programs were then further subdivided by subject area. Of the 52 programs listed for use by students in mathematics, social studies, and English, four were simulations, three were designed to teach programming; and two were aids for writing poems. The remaining 43 could best be described as drill-and-practice programs and games. In contrast, every program that respondents favored for their own professional use was a software utility or tool--word processor, database management system, program editor, graphics editor, spreadsheet program, or file manager. Why is there such a large gap between what passes as educational software and the type of software favored by teachers? If, in society as a whole, computers are used as a tool, why is this one of the most neglected uses in the classroom?

There are a number of reasons for the slow evolution of educational software. Historically, computers were first viewed as substitutes for doing more easily what had previously been done using older technologies. For example, in business the computer was first used to take over ongoing activities such as payroll management and accounting. Few business people actually used computers, and thus were slow to appreciate the full range of potential computer applications. It wasn't until the development of a new generation of office software tools in the mid-seventies that the situation in the business world began to change. Starting with VisiCalc--the first of the general-purpose spreadsheet programs--users of business computers began to see that, in addition to such traditional tasks as budgets and accounts, computers could make possible a range of new activities that were previously too expensive, too complicated, or too time consuming to be practical. The development and rapid integration into the work-

*Paper presented at the annual meetings of the American Educational Research Association, Montreal, Quebec, Canada, April 11-15, 1983.

place of spreadsheet programs, relational database management systems, and powerful word processors radically changed the way information was collected, manipulated, and transmitted in the office environment.

In education, we are beginning to see a similar trend. Until now, the primary use of computers has been to replicate what teachers have been doing with other technologies (e.g., workbooks, dittos, flashcards). Computer-assisted instruction (CAI)--using the computer as an automated workbook, like business payroll programs--is an obvious entry point. In some contexts--again like payroll programs--CAI programs have significant advantages over noncomputerized systems. However, the type of activity involved is essentially the same whether or not the computer is being used.

In the educational sector, we seem to be in approximately the same position as was the business world in the mid-seventies before VisiCalc. For the most part, schools are using computers--often begrudgingly--as automated workbooks or, to a lesser extent, to teach introductory programming. Thus the bulk of children's experiences with computers resembles those that were available in classrooms before computers.

How can we account for this relatively limited use of computers in the classroom? Why has there been no educational equivalent to VisiCalc that demonstrates the power of the computer as a multi-purpose tool? Part of the answer lies in the economic constraints on producing software exclusively for the educational market. However, not all the blame can be placed on the lack of financial resources to attract talented programmers and designers to the educational market. Two other factors have placed limits on the development of effective software tools for schools: (1) the need to restructure classrooms, classroom management techniques, and teaching philosophies to accommodate innovative uses of software; and (2) a lack of information for software designers and programmers concerning teachers' needs and the developmental capabilities of children of different ages. A cursory review of most educational programs reveals that the designer and/or programmer gave little consideration to the capabilities of the intended users. The reading level or conceptual requirements of many programs are frequently too high and require a herculean memory to keep track of what to do, how to do it, and when to do it.

Conceptual models that help teachers think about innovative uses of software in the classroom have also been slow in developing. Teachers and school personnel who use educational software face a chaotic situation. Often untrained, they lack a framework for easily and flexibly integrating the technology into classroom curricula and organization. They must rely on printed materials, manuals, and the software itself for information about computers in education. Currently, efforts are being made to evaluate software for teachers by various groups and educational computing magazines. However, because of their narrow focus on the behavioral and managerial objectives of individual classroom programs, these review efforts tend to perpetuate the narrow focus of the software being produced.

Software designers tend to concentrate on issues of appeal and individual usability, largely ignoring deeper concerns about learning, particularly in the classroom context. There is no body of research to which designers can turn to help them relate the design of educational software to the developmental factors involved in children's learning or to the context of classroom use. Further, the design of educational software is a new and varied craft. Producers range from individuals in "cottage industries" to large publishing houses. Cottage industry programmers often lack the educational background and/or programming experience and resources to adequately design and then test their programs. Publishers, faced with a diminishing school market for printed materials, are under similar constraints. Rather than devoting resources to the development of innovative software and doing the necessary formative research, they opt for a conservative strategy of producing software that complements--and looks very much like--their textbook series.

Given the restricted range of software commonly found in classrooms, what are students learning about computers? No matter how much content a particular drill-and-practice program gets across, running such a program does little to increase students' understanding of computers or their uses. No one type of software can possibly teach a student what a computer is or the full range of its use. What is needed is exposure to a wide range of different types of software and the different uses of the computer.

How are we to bridge the gap between what computers can do and what currently available software permits them to do? One answer is to adapt the powerful software from other fields, such as business, for use in the classroom. This is, in fact, being done by some energetic teachers, but several factors prevent this kind of adaptation from solving the problem of educational software. First, many of the better business programs operate with expensive equipment not commonly found in schools (e.g., 64K Apples with 80-column boards and two disk drives running the CP/M operating system). Second, the programs designed for business necessarily use examples in the manuals and in the screen prompts which pertain to business situations, thus making them seem less accessible to students. Further, the reading level required by these programs and their manuals tends to be high.

Another problem with many business programs is that they are designed for the heavy user and, thus, tend to include a large number of powerful but cryptic commands. Once mastered, these commands give the user tremendous flexibility, but they take time to learn and are easily forgotten if not used frequently. Many word processing systems suffer from this problem. For example, in one of the best selling word processors for the Apple, SHIFT CONTROL-Q is the command for deleting the word following the cursor. This is just one of some 130 commands that this program makes available to the user. This poses no problem in an office, where the program is used daily by a single typist. However, in the classroom, where students may only have one hour a week to use the word processor, this constitutes a serious hindrance.

A final problem with software tools designed for the business environment is that they include many features that are rarely, if ever, needed in the classroom (e.g., printing in multi-column formats, providing facilities for automatically calculating mortgage rates, or loan depreciation). The problem with such software tools is that they tend to be more difficult to use, take longer to learn, have more complicated manuals, and offer less comprehensive on-screen prompts because of the large number of commands.

What is needed are software tools designed specifically for students to help them do their required school tasks, and to prepare them to handle the powerful, specialized software tools that they will encounter in out-of-school contexts.

Software Tools for Schools

The need for more classroom software tools cannot be met by simply removing the frills from existing office software and then passing these stripped-down programs on to the schools. The realities of classroom life--the type of work required of students, the number of users per computer, the expertise of the teacher, and the constraints on student access to the computer--require programs designed specifically for this environment.

Many business programs for microcomputers are simply modified versions of programs that have been in use for decades on larger machines; educational software designers do not have this background to draw upon. Nevertheless, the domain of software tools designed for education is not entirely without representation. Researchers and curriculum designers around the country are beginning to make available the first generation of software tools designed for the classroom. For example, the Bank Street Writer (Broderbund Software and Scholastic, Inc.), Quill (Bolt Beranek and Newman, Inc.), and The Writers Assistant (Jim Levin, UCSD) are examples of language arts programs inspired by the electronic office of the future but designed for the classrooms of today. These programs share the basic design goals of ease of understanding and use, and of putting the student in control of the computer. Quill, for example, is a family of programs (word processor, electronic mail, message center, database) linked together as a single system through which students can write and communicate. Although Quill and other classroom software tools differ from their business counterparts in some respects (e.g., the size of files they can handle or the number of records they can maintain), they retain the power, ease of use, and flexibility which have made software tools so popular in other segments of society.

The lesson that the developers of these programs have learned from successful business tools is the critical interaction between software and the context in which it is to be used. In order for software to be successfully incorporated into a work or educational environment, it must be recognizable and usable as something fitting the work needs of its intended users. To be maximally effective, the exploration and exploitation of the software's potential must yield something more powerful than numbers or words printed neatly on a page--namely, new ways of thinking about, organizing, and communicating information.

Developing Educational Software Tools: The Case of the Bank Street Writer

What are the design decisions that must be taken into account when developing software for the educational sector? How should software designers think about the special needs of children and teachers in the context of the classroom? While these questions cannot be answered as yet, we can provide some preliminary ideas based on our experience in developing the Bank Street Writer, a classroom word processor jointly produced by the Center for Children and Technology at Bank Street College, and Intentional Education, the publishers of Classroom Computer News.

The decision to develop a word processor specifically for children grew out of research on children's revision strategies that was being conducted at Bank Street College. Researchers were interested in how working with a word processor would affect the number and types of revisions students would make while writing. As part of this research, a number of word processors for Apple and Atari computers were carefully reviewed. This process included critiques by student-user panels as well as by experienced adult users. It was found that, although students enjoyed writing with the computer and willingly put up with the clumsiest editing features, each of the programs reviewed had serious flaws which, even with some modifications, would make them too difficult for classroom use. The programs took too long to learn, required the memorization of too many commands or rules, had difficult-to-read screen displays, and made editing so clumsy that the whole point of using the word processor for revising text was lost.

The decision to create a word processor for the classroom was based on these findings and on the growing realization that a word processor designed specifically for the young writer would have to be very different from those designed for office use. The word processors used by secretaries to enter, revise, and produce someone else's work are complex and emphasize functions necessary for the formatting and production of professional documents. A word processor designed for creating text (i.e., one geared to the needs of the writer, not the production editor) must place as few obstacles as possible between the writer and the process of creating and manipulating a text.

After making the decision to create a tool that would meet the needs of the young writer, we had to settle on a design process. A fundamental belief shared by project staff was that no one person had the expertise required to produce a good educational software tool. Thus, the next and most important step was to assemble a design team consisting of researchers interested in writing, classroom teachers, curriculum designers, professional writers, and the programmer who was actually to implement the design. The concept of a design team for creating software, while not new, is not yet typical. Although groups of programmers or engineers often work jointly on a program, involving researchers, educators, and end users in the design and validation process is rarely done. Project staff felt that the robustness of the final design would depend on the pooled sensibilities of the interested parties.

The design team began by establishing a set of design principles: no control character commands would be used; there would be comprehensive screen prompts; the display would show upper- and lowercase letters; and no hardware modifications or additions would be needed in order for the program to run on a typical school computer system. This was followed by identifying those functions that the program would carry out and those that it would not. For example, it was decided that students might reasonably need a MOVE command to relocate blocks of text or a FIND command to locate a particular word in the text, but would have little use for or understanding of macro commands that, at a single keystroke, can execute a sequence of user-defined instructions. Similarly, commands for creating indexes, multi-column printouts, and setting line traps were judged not to be features that children would use in their typical writing tasks.

Once the initial design had been agreed upon, the programmer put together the prototype, which the design team then critiqued, refined, and field tested with students and teachers in several settings. The field-test results led to further discussions and refinements by the design team; for example, it became clear that the names for some of the commands were misleading and needed to be changed. More serious changes involved redesigning the FIND and REPLACE commands, and working out a way for students to get professional-looking printouts without using formatting commands. After repeating this test-and-refine loop several times, the program was considered ready for general use and was released to the public. We are now in the process of collecting feedback from the many users who have purchased the program, and are using their experiences to improve new versions of the Bank Street Writer and to help in the design of future classroom software tools.

Our major conclusions about the design of effective educational software tools can be summarized as follows:

1. Students, like adults, are purpose-oriented. Most elect to use a piece of software because they want it to help them with a specific job. They do not wish to spend large blocks of time learning a program before they are able to do something interesting with it. Software tools, therefore, should provide easy entry into the program, and more sophisticated features can be added if and when needed.

2. Students do not like to read complicated manuals in order to master a program. Ideally, a manual is not needed; the program itself should provide the prompts and messages necessary for its use. Any accompanying material should be short, function-oriented, and should anticipate students' needs and the problems they are likely to encounter. Separate manuals for teachers and students may be considered if the program is designed so that the teacher can select options for students.

3. Manuals are not novels. Few people read them from cover to cover or in order of chapter. Thus, it is important for critical information to appear throughout the manual to ensure that the user will see it sooner or later. A better solution is to incorporate flexible on-line help files into the program which can be accessed from within the program without interfering with the user's work.

4. Neither students nor adults like to read lengthy instructions on the computer screen. Prompts which help children to use the program are very important but should be kept simple; they should inform students about the program's current status and the options available at that time. In addition to serving as memory cues for the program's functions, prompts can help in the development of better writing skills by directing the user's attention to such stylistic considerations as audience awareness, parallel construction, or the use of suspense as a plot device.

5. Students should be able to know at all times where they are in the program and how to get back to a particular screen or function but many programs do not provide this kind of assistance. Using the program should not be a test of the student's power of deductive reasoning or memory capacity.

6. Students should be able to use commands easily and to enter their choices readily. For example, each command should require one keystroke; multiple keystrokes to achieve one result are confusing for children. <u>How</u> features are implemented is as critical as <u>what</u> features are provided.

7. The program should let students know that something has happened after a command has been entered. Children become confused if their input seems to have no effect or to "disappear." For example, when a file is SAVED on a disk, the program should inform the user that the operation has been successfully completed.

8. Students sometimes make errors when using a program. A classroom software tool must have built-in safeguards against program and disk errors. This means the program should not just stop if an inappropriate command is entered; rather, an error message should appear indicating what was wrong and how to correct it.

9. Good software must be sensitive to the user's theories of how the computer and its associated parts function and interact. For example, students often get confused about where the program and their text is stored at any given moment. The distinctions between the computer's volatile RAM memory, virtual disk memory, and perma-

nent disk memory are hard to grasp. A program that cannot eliminate the need to memorize these distinctions must be defaulted to automatically check that data is periodically saved properly, and to give ample warning before a student can erase or forget to save data.

10. In many classrooms, students have to share data disks, and a single program disk has to serve several machines. Therefore, programs should load completely into memory so the program disk can be removed and used to load the program into another machine. This feature also lessens the chance of the system's "crashing" in the middle of a session. Since several students may be using the same data disk, it is important to have some protection scheme (e.g., passwords) so that files can be saved without the risk of other students accidentally (or purposely) erasing or modifying someone else's work.

11. Students will not need all the features that might be included in a given tool. It does not follow that, because a computer can perform a certain function, this function must automatically be included in the program. Software developers are continually faced with the problem of making a flexible and powerful program which can do its primary tasks simply and clearly. The design team must determine which features are most important for student users and eliminate those that are unnecessary. For example, when the Bank Street Writer was developed, it was decided that students would not need the sophisticated and difficult-to-use text formatting features so important for office word processors.

12. Designing a tool that can be used effectively by young children can be a good way of making a program that adults will also find useful. Children do not need stripped-down versions of adult programs but, rather, programs which facilitate, as elegantly as possible, the execution of certain kinds of work. Thus, a well-designed program that promotes children's writing should facilitate the writing of adult users.

These principles provide a framework for thinking about the design of classroom software tools. As the realization grows that CAI and programming do not exhaust the possibilities for computer use in schools, teachers will increasingly be attracted to those programs that can turn their computers into powerful tools, thereby serving a diversity of curriculum goals. We look forward to seeing, in the near future, the emergence of a wide range of classroom software tools to parallel the many uses for the computer already in place in other segments of our society.