DOCUMENT RESUME

ED 249 597 EA 017 122

AUTHOR Walk., Decker F.
TITLE The Software Problem.
PUB DATE Jun 84
NOTE 7p.; In "Making Cur Schools More Effective: Proceedings of Three State Conferences." See EA 017 101.
PUB TYPE Speeche./Conference Papers (150) -- Viewpoints (120)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Computer Assisted Instruction; *Computer Software; Curriculum Development; *Economic Change; *Economic Factors; Educational Economics; *Educational Needs; Educational Trends; Elementary Secondary Education; Programing
IDENTIFIERS *Market Analysis

ABSTRACT

This paper addresses the reasons that it is difficult to find good educational software and proposes measures for coping with this problem. The fundamental problem is a shortange of educational software that can be used.as a major part of the teaching of academic subjects in elementary and secondary schools--a shortage that is both the effect and cause of insufficient market demand for more diversified educational software. Related problems include (1) the time and cost of designing and coding programs, (2) machine incompatibility, (3) software piracy, (4) locating and reviewing software, (5) competition by manufacturers for the more lucrative home market rather than for schools, and (6) problems in integrating software into a classroom's other activities. Because the software problem is primarily economic, the principal solution is to provide economic incentives for educational software development. These include government and private funding of software projects, collaborative contracts between software firms and educators, and wise purchasing decisions to help shape the course of production and innovation. Curriculum design should incorporate highly adaptable generalized programs, and teachers can develop their own educational software. The problem will be mitigated only as more organizations invest more in developing good educational software. (TE)

THE SOFTWARE PROBLEM

Decker F. Walker
School of Education
Stanford University

When the time comes to use computers for something beyond computer programming and computer literacy, the software problem looms. There seems to be a great deal of software: simply wading through all the titles searching for what you want can be a day's work. But try finding some software to teach exactly what you need taught and you will, at least nine times out of ten, encounter the software problem.

Why is it so difficult to find good educational software? Is there anything a teacher or school administrator can do to alleviate the problem? Is it likely to get better in a few years? These are the questions I will consider in this section. In considering them I will need to begin with the more basic issue of how computers are used in schools.

Computers in Schools: Varied Patterns of Use

Computers can play a variety of roles in education, ranging from the most marginal of roles--as a supplementary optional activity for a few students, to the major role as a "teacher" of a course. Much of the excitement about computers in education is attached to the idea of the computer as a "teacher" in its own right--as a Socratic tutor, as a magnificent diagnostic device locating and remedying students' misunderstandings, or as the ultimate audio-visual device branching students through the Library of Congress on personalized learning paths. But the present reality is that computers play mainly a marginal role in schools except in computer programming and computer literacy classes and such vocational courses as typing, accounting, and electronics.

The software problem is relatively mild in these applications because teachers can use the software developed for more general purposes, such as operating systems, languages, word processors, accounting programs, and the like. But when we turn to important applications of computers to mainstream academic courses such as English, math, science, social studies and languages we find the software problem much more severe. We also find, and it is no coincidence, that use of computers as a major part of the teaching of these academic subjects is quite rare today.

That the software problem is most severe in mainline academic subjects is an important clue to some of the origins of the problem. In those educational applications where computers are now being widely used--computer literacy, programming, vocational applications--generally only a single piece of software is needed, an operating system, a language, or a word processor. In the teaching of an academic subject like algebra, however, perhaps as many as a dozen or more substantial programs will be needed if the computer is to be useful over the entire course. If we multiply the number of subjects taught (perhaps an average of 8 per year) by the number of grade levels (12) by the number of programs needed per course (say 10) we see that nearly 1,000 programs are needed simply to cover the public school academic curriculum with only one program per topic.

So, even though software seems to be flooding the market--our files at Stanford include over 100 catalogs of educational software with more than 3,000 titles--coverage of the software needs in academic subjects remains spotty. When you consider that most of the items in the catalogs are concentrated in a few subjects and topics (elementary math drill and practice, spelling, and computer literacy), it is easy to appreciate the enormous variety of the demand for software. This variety means that the market for any single piece of software filling only one of these thousands of niches will be smaller than the market for more generally useable programs. Until the number of computer-using teachers in the various subjects increases substantially, the market for software in those subjects is too limited to justify an investment in producing it. And, so long as the selection of software is limited, many schools and teachers will be reluctant to use computers in their teaching. This is the familiar Catch 22 situation that confronts any innovation, but the fragmentation of the market into so many niches makes the problem more severe in the case of substantial applications of computers to mainline academic subjects.

Good Educational Software

Computers have been called "chameleons in the classroom" because they can be used in so many different ways. Computers used for drill and practice with individual students seem so different from the machines used as an "electronic blackboard" to present animated geometry diagrams or from the machines used by a small group of students in a simulation game. The differences are produced by the software: the computer may well be the same machine in all these applications.

The qualities that make a computer work well for one of these educational uses may not necessarily be desirable in the others. And this is another source of the software problem: varied criteria. A piece of software that does a good job of teaching arithmetic facts through drill and practice will not satisfy educators who want programs that develop understanding of number concepts. A program that entertains and motivates students with color graphics and animation will please those whose educational philosophy is child-centered and displease those whose philosophy is more subject-centered. Such diversity of criteria further reduces the likelihood of finding software that will be generally regarded as good and increases the number of niches in an already fragmented market.

Finally, we must recognize how high are the standards typically used to judge educational software. Few dispute that computer programs can teach number facts, but we also know that traditional methods such as flashcards, can do the same job and much more cheaply. Computers, being more costly, must accomplish more than traditional methods if their use is to be justified economically. By extension of this line of reasoning, ways must be found to use computers to teach the most difficult concepts and skills, those which substantial numbers of children now fail to learn using traditional methods. To develop programs that achieve these high standards is not an easy task. We certainly cannot expect that anyone should be able simply to sit down and write such programs. They require thorough analysis, deep thought, and inspired design.

Current Dimensions of the Software Problem

The fundamental problem is a shortage of educational software that can be used as a major part of the teaching of academic subjects in elementary and secondary schools. The number and variety of programs needed to alleviate this shortage is large, but, as the saying goes, "you ain't seen nothin' yet." We have yet to consider several other aspects of the problem that make it larger and more severe than it seems so far. The aspects that follow are presented in no particular order.

1. Development time and cost. The best estimates of the time required to design and code a computer program range from 100 to 300 hours per hour of running time. This does not include the time needed to think up the program ideas. This translates into a development cost for a program that students might use for one hour of between $2,000 and $100,000, depending on its sophistication and complexity. By contrast, to produce text material to occupy a student for an hour is a matter of a few hundred dollars at most. And remember that the market for the software is limited by the number of machines available and the large number of small niches in the market, much greater limitations than apply to text materials.

2. Machine incompatibility. A new form of Murphy's law: The program you want is only available for a machine you don't have.

3. Software piracy. Software manufacturers are reluctant to invest in the development of products that will be copied at no charge by the customer. If one can sell only one copy of a program per school, the price necessary to recover the investment must be large, between $300 and $500 per copy. This, obviously, makes it prohibitively expensive for a school to buy enough copies to supply one for each computer and therefore ensures either that the software will only be used as a supplement or that it will be illegally copied.

4. Locating and reviewing software. Even when good software exists, finding it and verifying that it is good are nontrivial problems. Indexes are beginning to appear that list software by subject, grade, and other useful properties, but at this moment coverage of such indexes is spotty. A number of journals publish reviews of software, but finding a review of the program you have in mind remains difficult. What we need are specialized publications that review programs in a small area with particular reference to their usefulness in the classroom. Again, we are confronted with the problem of a plethora of small niches which make it uneconomic to provide reviews to such a small audience.

5. Competition for the home market, rather than the school. The number of installed machines in homes far exceeds the number in schools, and individuals buy a total dollar volume of software several times greater than schools. Software manufacturers can therefore sell to a larger market by producing for the home. And most of them do. This means that software is designed primarily for conditions in the home--one student per computer, unsupervised use, episodic use with little extended continuity in the development of skills and ideas.

6. Problems in integrating software into the classroom's other activities. Even a well-designed piece of software will not fit exactly into a given teacher's plans. Adjustments must be made to accommodate it. If the software is not modifiable, then all the adjustments must be made elsewhere, and there are limits to a teacher's willingness to tailor everything else to one program. And when a teacher uses several programs in the course of a year, each of which requires a different set of adjustments, the problem may become insurmountable. Examples include the spelling program whose words do not match the teacher's goals, the math program which introduces skills in a different sequence from the school's curriculum, and the science program which uses a different notation from that in the textbook.

All these difficulties translate into a higher cost to provide the software needed. The cost of equipping a single course in one school with enough software to be used one hour per week for 30 weeks in a school year, assuming appropriate programs were available at today's typical price of $50 per diskette, and an optimistic 'playing time' of three hours per diskette, and one diskette for each three students in a thirty-student class, comes to $5,000. This figure is too expensive by a factor of ten. So long as costs are this high, the market in schools will be thin.

What To Do?

What can be done today to overcome the software problem? The software problem manifests itself as an economic problem, even though not all of its causes are economic. The home market for educational software programs will likely continue to be bigger and richer than the school market, and therefore software companies will continue to produce for that market. Eventually, competition for that market will make the smaller niches in the school market relatively more attractive, and we will then see more production of software specifically for schools. In the meantime, however, the home market is far from saturated, so the present situation is likely to continue for some time.

Only large-scale actions would change this economic situation substantially. If the government and private foundations could be persuaded to finance dozens of software projects in education, that would make a dent in the problem. If districts formed consortia and invested their own funds in the development of software, that would have a significant impact. Million dollar contracts between software development houses and school districts to develop software collaboratively and share royalties would have an effect. And the simple growth of a school market for software will, in itself, stimulate more and better software.

Progress in this fundamental aspect of the problem requires growth of investment or expenditures or both. If this growth fails to come or comes slowly, the software problem will not improve, regardless of anybody's good intentions or hard work.

In the present thin market, buyers' decisions have an immediate and powerful shaping effect on producers. Those products that sell will be widely imitated, while those that do not will rapidly disappear from the catalogs. Schools can influence the future direction of growth in software by being discriminating buyers. Buy only programs that give you the most for your money. One measure of the value of a piece of software is the number of student-hours of use per dollar of cost. This figure ought to be computed in reviews of software prior to every purchase. Another important quantitative indicator is the extent of your curriculum covered by a program. One that is useful in only 1% of a year's classes is less valuable than one useful in 10% of classes.

What actions can an individual school or district take to cope with the software problem? One thing that should be done is old-fashioned curriculum development. Scope and sequence charts are needed showing just where what types of computer programs can be used and teacher's guides showing how they can be integrated with the other ingredients of a good course.

The problem can be eased by extensive use of tool-type programs and modifiable programs. The Music Construction Set is a program that transforms a computer into a composer's typewriter. A staff and various symbols are displayed on the screen and these can be moved around with keyboard commands or a light pen to compose music which can then be played by the computer with the press of a button. Such a program can be used throughout the year in a music class. A spelling program which permits teachers to enter their own words is much more valuable than one with a fixed word list. Insist on programs that can be used as tools by teachers and students, programs that can be tailored to suit your curriculum.

One type of educational software that is little known in this country but widely discussed in Europe and Japan can be used to great effect in academic classrooms. These programs are called "electronic blackboard" programs. One such program, Quadrilaterals, is published by Reader's Digest. A teacher uses this program on a single computer at the front of the classroom, with a screen large enough for all students to see. Using game paddles, the teacher is free to walk around the room while controlling the display. The teacher can choose to have text displayed or only diagrams. Questions can be posed for class discussion and then the animation powers of the program used to show the answer on the diagrams. The program is used very much like a chalkboard by the teacher, so that no extensive inservice is necessary to prepare teachers to use it. Such programs can be extremely cost-effective ways to use computers in the teaching of academic subjects.

Teachers can create their own educational software. To do this 'from scratch' in Basic or assembly language is a difficult and time-consuming activity that cannot be expected from teachers working full time. But, using an authoring system such as Pilot, teachers can develop lessons in only slightly more time than it takes to develop ditto masters or overhead projection sheets. However, to develop software that is truly interactive and that accomplishes things conventional methods cannot remains a high art, difficult and time-consuming. It might be reasonable to expect a talented, dedicated teacher working with an authoring system after school, weekends, and holidays, to produce two or three hours' worth of such programs in the course of a year, but not more. Unless your school has an unusual concentration of these rare birds, it is unwise to rely upon teacher-made computer software for a major part of your courseware. If you are determined to rely on teacher-made software, you might consider contracting with the most able teacher-developers to spend a substantial portion of their time for a year or so developing software. You might even consider entering into a consortium with neighboring districts to pool the talents of your teacher-developers.

It is possible to lease or purchase a complete set of integrated software designed for school use. Computer Curriculum Corporation, for example, has complete computer-administered courses in most of the subjects of elementary and secondary schools. Many publishers of basal texts for the elementary school also offer computer software designed t accompany and enhance their text materials. Control Data Corporation's PLATO system offers a good selection of software for most school subjects. These integrated software systems are expensive, and they are not tailorable to an individual school's or teacher's needs, but they may be more cost-effective than assembling your own software from catalogs or developing teacher-made materials in many instances.

Over a somewhat longer period and on a larger scale, you might contemplate entering into collaborative arrangements with software developers to work on the most pressing software development needs. Consortia of districts contracting with private developers and involving local teachers in the development process can be a powerful development strategy. Collaboration is also possible with professional associations. Such initiatives put your school into the software development business and this might be difficult to arrange with your board, but they give you more control over the software than you get any other way.

In summary, the software problem is serious and can be traced to some fundamental economic causes that are not easily overcome, but there are constructive ways to cope with the problem if you are willing and able to invest the money, time, energy, and initiative. The problem will not go away in the foreseeable future, in any event, and it will only get substantially better as more organizations invest more in developing good edutational software.