

**The High School Computer Center:
Educational Goals in Theory and Practice**

Copyright © 1985

Brian Keith Harvey

Acknowledgements

I am grateful to the teachers and students whom I interviewed, and whose classes I visited, at the five schools studied in this work. They gave me generous portions of their preparation and lunch periods, as well as tolerating the distraction of my presence in classes.

All of the members of my thesis committee have given me both good advice and patience in the face of my spurts of furious activity and lengthy inactivity. They are Professor Richard M. White, chairman; Dr. Joan I. Heller; and Professor James L. Jarrett. Thanks also to Professor Richard Fateman and Professor Charles Woodson, who served on my qualifying examination and proposal review committees.

All of my fellow students in the Graduate Group in Science and Mathematics Education have been extremely supportive; I would particularly like to thank Batya Friedman for her critiques of early drafts of my proposal and of this thesis.

Table of Contents

1. Introduction	1
2. School A	
Description of the Facility	9
Courses Offered	10
Classroom Observations	12
Interviews	19
Summary and Conclusions	21
3. School B	
Description of the Facility	24
Courses Offered	24
Classroom Observations	25
Interviews	28
Summary and Conclusions	34
4. School C	
Description of the Facility	37
Courses Offered	38
Classroom Observations	38
Interviews	41
Summary and Conclusions	44
5. School D	
Description of the Facility	46
Courses Offered	46
Classroom Observations	49
Interviews	53
Summary and Conclusions	58
6. School E	
Description of the Facility	59
Courses Offered	60
Classroom Observations	63
Interviews	73
Summary and Conclusions	81
7. Educational Goals: John Dewey	85
8. Educational Goals: Martin Buber	95
9. Educational Goals: Frank Oppenheimer	106
10. Educational Goals: Seymour Papert	112
11. Conclusions	118

Appendices

A. Stop Saying “Computer Literacy”!	129
B. Computer Hacking and Ethics	136
C. Course Materials	144
References	181

List of Illustrations

School A

Figure 1. Computer Science course offerings	11
Figure 2. Structured programming	14

School B

Figure 3. Philosophy and goals	29
Figure 4. Curriculum topics	31

School D

Figure 5. Introduction to Computers	47
Figure 6. Computer Programming	48
Figure 7. Programming problem	52

School E

Figure 8. Sample curriculum guide	61
Figure 9. Personal Computing Course Introduction	66
Figure 10. Computer Science course introduction	71

School A

Appendix C.1. Computer Discovery curriculum	145
Appendix C.2. Variables unit	146

School B

Appendix C.3. Bulletin board messages	149
Appendix C.4. Bulletin board map	152
Appendix C.5. Bulletin board commands	153

School D

Appendix C.6. Review test	154
Appendix C.7. Programming problem	155

School E

Appendix C.8. Introduction to Word Processing	156
Appendix C.9. Typesetting and Typesetting Lab	157
Appendix C.10. Personal Computing and You	158
Appendix C.11. Computers in Mathematics	160
Appendix C.12. Advanced Computer Science	161
Appendix C.13. Behavior Policies	162
Appendix C.14. Personal Computing worksheet	164
Appendix C.15. Personal Computing programming assignment	168
Appendix C.16. Non-programming enrichment projects	170
Appendix C.17. Programming enrichment projects	171
Appendix C.18. Computers in Mathematics test	172
Appendix C.19. Computer Science programming assignment	175
Appendix C.20. Computer Science programming assignment	176
Appendix C.21. Computer Science homework assignment	177
Appendix C.22. Advanced Computer Science introduction	178
Appendix C.23. Advanced Computer Science curriculum schedule	180

**The High School Computer Center:
Educational Goals in Theory and Practice**

Copyright © 1985

Brian Keith Harvey

Introduction

The computer is the latest piece of technology to appear in classrooms. The past decade has seen exponential growth in the number of computers in schools. Even this sudden growth has not been enough to satisfy some observers; as part of the wave of reports critical of American education, there have been several calls for further intensification of computer-based education:

- The National Commission on Excellence in Education issues a report, *A Nation at Risk*, advocating that every high school student be required to take a semester course in “computer science.”
- In an article typical of ones that appear in magazines every month, Dr. Robert Tinker begins with the remark, “There seems to be unanimous agreement that computer literacy should be an important part of every curriculum from kindergarten through college and that this should involve not only teaching about computers but teaching how to program.”
- A House Science and Technology subcommittee holds hearings on the educational use of computers; its chairman, Rep. Albert Gore Jr. (D-Tenn.) says, “Computers hold the potential to tremendously enhance the quality of education in America. But I am concerned that our educational system is not adequately prepared for this tidal wave of technology.”

These political and educational leaders are responding to a technological imperative, not an educational one. Their remarks do not address educational goals that make the computers necessary. Instead, computers are self-justifying: because they exist, we must have them in the schools. There is a feeling of panic in these remarks.

There have been educational fads before, but the computer is a more complex one than most. In the past, there have been new pieces of educational technology, such as the language laboratory, designed to help meet an already well-defined goal. Such innovations do not raise philosophical questions, but merely the pragmatic question, “Does it work?” There have also been new curricula designed around well-articulated new goals, like the New Math programs designed to introduce children to formal mathematical thinking as a first step in training a new generation of scientists. For these innovations, in addition to the pragmatic question, there is the philosophical one: “Is this really something we

want to do?” But the computer is harder to pin down because it shares attributes of both of these categories. The computer can be a tool to aid in pursuing existing goals, as when it is used for drill in arithmetic; it creates new goals as an object of study in itself, as in computer programming courses; it can also change the terms of discourse about existing issues, as in the elevation of “problem solving,” already an educational goal before computers, to a new position of importance. In this case, before we can pass judgment on the goals of educational computing, we must first ask what those goals *are*. Much activity in computer-based education has not asked that question.

Some educational leaders have noticed this lack of clear goals. Ernest L. Boyer, the President of the Carnegie Foundation for the Advancement of Teaching and author of its recent report about high schools (Boyer, 1983), says, “We’re buying [computers] now and planning later... I think we need to blow the whistle on this and find out what the computers can and cannot do.”

In this study, I ask two questions: What goals are high schools actually pursuing in their use of computers? What other goals might computers fruitfully serve?

What Are Schools Doing with Computers?

There has been some study of the ways in which schools are using computers. One important research project in this area is the National Survey of School Uses of Microcomputers, conducted in 1983 by the Center for Social Organization of Schools, Johns Hopkins University (Becker, 1983). This survey explored microcomputer use in elementary as well as secondary schools. Among the results is that 85% of all high schools in the survey have one or more microcomputers. In the typical (median) high school, computers are used by 77 students each week, or 13% of the student body. Each student in the typical school gets 45 minutes of computer time per week.

Of the high school teachers who responded, the largest number (85%) reported that they use computers for a course introducing students to the use of the computer itself; 76% use computers for instruction in computer programming. The third most common use, drill and practice, was reported by only 31% of the teachers. Other uses reported by 19%

to 29% of the teachers are business or vocational education; programs to solve problems in other areas; demonstrations, labs, or simulations; special education; and recreational games. Word processing was reported by the smallest number of teachers, 7% of the total.

This extensive survey also reported results about such things as who made the purchasing decisions, where computers are located, how many students work together on a computer, and so on. The results I have summarized here are the ones most relevant to the present study.

Why Are Goals Important?

The Johns Hopkins study sheds much light on questions about how schools are using computers. I am asking a different question: *why* are they doing what they do? The first question is essentially one of instructional methodology; the second is a question in applied educational philosophy.

Many educators are impatient with philosophical questions. They prefer to ask pragmatic questions with easily verifiable answers. Will this new curriculum increase test scores? Would starting schooling a year earlier raise or lower the high school dropout rate? The implicit assumption behind such questions is that the prior issue of *why* we should increase scores, or discourage dropouts, is self-evident or already settled.

The past history of educational computing provides a sharp example of why it is important to raise goal questions explicitly. The most common use of computers in education is not, in fact, any of the ones listed in the Johns Hopkins study. It is a use that predates the invention of the microcomputer: the automatic grading of standardized, multiple-choice examinations. This practice was once an innovation. It was introduced as an efficiency measure, not as a new direction requiring new philosophical thinking. Its effects, however, have had important implications for educational goals.

The most profound effect of standardized tests has been on educational epistemology, the definition of what it means to know something. Traditionally, educators have recognized many kinds of knowledge. There is the knowledge of specific facts (“knowing that”), the knowledge of skills (“knowing how”), the understanding of relationships among facts, and so

on. Corresponding to these types of knowledge are various methods for testing a student's knowledge: factual questions, essay questions, construction projects, class discussions, and others. Computerized testing has not eliminated this range, but it has given a much greater emphasis to factual questions than in the past. This implies a corresponding emphasis on factual *knowledge* as the goal of schooling. Students pick up this message, whether or not their teachers tell them explicitly. Even within the category of factual questions, computerized grading makes such questions more of an all-or-nothing affair. There is a single right answer. In the past, when such questions were graded by human teachers, it was possible to get partial credit for a partly correct answer, or even to convince the teacher to accept an alternative point of view as fully acceptable. The change to standardized tests is not just a pragmatic one; it has changed the meaning of the words "question" and "answer" in our culture.

A second change brought about by standardized testing has been in the area of public policy in education. Standardization means centralization, and in fact the centralization of testing has left a private company, the Educational Testing Service, as a sort of *de facto* governing board of every secondary school district in the United States. The elected officials who are supposed to set curriculum policy must agree to teach the implied curriculum of the Achievement Tests in order to get their students admitted into college. This shift toward a uniform national curriculum is not necessarily a bad thing; it is possible to argue that the current situation is an improvement over the days when a state legislature could declare pi equal to four. Good or bad, however, this change was *accidental*. A new technology was introduced without consideration of its effect on educational goals.

The inventors of the computer-graded examination were not thinking about epistemology or about public policy in curriculum design. Their concern was with saving teachers' time by eliminating a boring task. Today, most educators deplore the increased importance of factual questions with a single right answer, but this new epistemology is considered inevitable, a necessary evil. The idea behind this dissertation is that the second generation of educational computing may have similarly unintended consequences. These consequences may be good or bad, but they are more likely to be good if the philosophical issues are raised *explicitly* before any particular style of microcomputer use becomes "inevitable."

Organization of the Study

This study has two parts. First I present the results of visits to computer centers at five public high schools. At each school I observed classes and interviewed teachers and students about their goals. The purpose of this part of the study is to give a picture of the goals of computer education as it is currently practiced. Goals are sometimes explicitly presented in written plans for computer education in a school district. The same or different goals are described by teachers in the interviews. A third viewpoint from which to understand the goals of a computer education program is to infer goals from the classroom practices observed.

The second part of the study is a review of educational goals in the literature, both from philosophers of education and from practitioners. The purpose of this review is to provide a contrasting background against which to interpret the current practice. The purpose is not to imply, much less to prove, that any of the writers reviewed here has “the correct goals.” Goal questions do not have right answers. These writers provide examples of thinking about education in a way that places goal questions first, instead of beginning with a technology.

In the conclusion, I hope to show that educational goals interact in important ways with actual practice in high school computer centers. It will be seen that the schools’ explicitly stated goals do *not* have strong implications for their practices, but that there are implicit goal decisions that do have an effect. I then give an example of how a high school computer center might be designed that would take seriously some of the goals suggested in the philosophical literature.

Methodology

To select the schools for the study, I began by asking the computer specialists from county Offices of Education in the Bay area to recommend high schools that, in their opinion, were using computers in exciting ways. My purpose was to study schools that are considered above average in educational computing, because I expected to find the greatest depth of thinking about goals there. It would not be interesting merely to prove

that schools with bad practices also have bad ideas.

From the dozen schools that were recommended to me, I narrowed the list to five by calling the teacher in charge of the computer facility at each school and asking for a brief summary of what they do with computers. From these capsule descriptions, I tried to choose schools that were as different as possible from each other. For example, one of the schools uses its computers entirely for vocational education, while the others use them for more academic purposes. I chose another school because of the electronic bulletin board that it operates. I then arranged to visit the selected schools.

I chose to concentrate on schools with computers concentrated in a special room set aside for the purpose, rather than on schools with small numbers of computers in each of several classrooms. One reason is that the concentrated computer laboratory is more typical at the high school level (although not in the elementary schools); another is that I expected a teacher who is a computer specialist to have thought more deeply about the goals of computer-based education than a teacher for whom the computer is merely one educational tool among many.

Some schools have more than one computer facility, with each run by a different academic department for a different purpose. Often the teachers in charge of these separate labs have very little communication among themselves. I chose to concentrate primarily on facilities in which the computer is an object of study, rather than a tool for the study of something else; again, this is the most common use of computers in high schools, and the one for which I expected to find the deepest thought about the underlying goals.

I spent between one and two days at each school. I tried to observe each computer course at least once, and to see both a lecture session and a lab session if the course was divided in that way. I conducted formal interviews with teachers during their lunch or preparation periods; I also conducted short, informal interviews with students before or after class, or while they were working independently. The formal interviews began with a uniform list of questions, but I also asked questions that grew specifically out of my observations at each school. Here is the list of questions:

What are your most important goals in computer-based education?
How do these fit in with your educational goals in general?

[For each goal listed:] How is goal X reflected in your daily activities?

How is goal X reflected in the structure of the computer center?

[When relevant:] How is the computer particularly appropriate for pursuing goal X?

Are your students aware that X is a goal for you?

What are your students' most important goals in computer-based education?

For the short student interviews, I asked two standard questions: "Why are you taking a computer course?" "Why do you think the school teaches a computer course?" Other questions were added when appropriate. For example, at schools with parallel (not sequential) programming courses in both BASIC and Pascal I asked "How did you decide to take this course instead of X?"

The schools I visited are all good schools, with good computer education programs. My expectation before conducting the study was that even the best computer facilities would have some weaknesses related to unclarity about educational goals. That expectation was correct, and so all of the reports of school visits have a partly negative tone. The reader should remember that my purpose is to point out directions for possible improvement in computer education everywhere, rather than to evaluate the merits of these particular schools in a comprehensive way.

What Is a Goal?

One of the difficulties I had in interviewing teachers about goals for computer education is the lack of precision in the definition of the word "goal." The most common answers referred to curriculum objectives: "I want my students to learn how to use a word processor," for example. Such objectives are tabulated in the Johns Hopkins study.

What I hope to illuminate is a deeper question: *Why* should students learn word processing? Is it to prepare them for a job? Is it to motivate the study of written English? Is it to overcome their fear of computers? Is this a goal for every student, or just for those who show a special interest in it?

The Philosophical Study

One way to get a vantage point from which to examine the current practice in computer education is to consider the history of educational goals outside of computer education. In the second part of this study, I review the work of four people who have commented on educational goals.

Two of the four are philosophers. I have not tried to include every possible educational philosophy. My purpose is not to point to some goal or list of goals and say, “This is what computer educators should seek!” Instead I want to present a sample of different directions, and in each case try to work out what it would mean for *computer* education to follow that road. My expectation is that different readers will find different philosophies appealing; some may prefer an approach not represented here at all. The point is that computer education, like any form of education, should be guided by *some* coherent approach. John Dewey was a pragmatist, a champion of science as the prototype of seeking truth, and a specialist in the philosophy of education. Martin Buber was a philosopher of religion, a mystic, and concerned with education as part of his broader concern with ethics and human relationships.

The third writer in this survey is not a philosopher, but an educational practitioner. Frank Oppenheimer was a college teacher and a high school teacher before he became known as the founder of the Exploratorium, a unique science museum with a well-articulated set of goals. Some of Oppenheimer’s goals are specifically related to museums, but others apply to education generally. I include him partly to show that thinking philosophically about goals is not an activity limited to philosophers and irrelevant to practice.

The fourth writer, Seymour Papert, is both a philosopher and a practitioner. Unlike the others, he writes directly and specifically about computers in education. He is not the only educator who combines practical understanding of computers with sophistication about goals, but he is an important example. It is not part of my conclusion that everyone should agree with Papert’s ideas, but I do believe that every computer-using educator should take him as a model of the depth of thought that is required of us.

School A

Description of the Facility

This is an urban high school, serving about 3000 students of mixed races and classes. The Computer Science Department offers classes in two computer laboratories and one regular classroom.

One of the laboratories contains 17 Commodore Pets on one side of the room, with face-front classroom seating on the other side. This room is shared with the Business Department, which offers a word processing course. The Computer Science Department uses the room for a course called Computer Discovery. The curriculum for this course includes “on-line” and “off-line” topics in each week; students spend some time each week in lectures, and some time using the Pets. On-line topics start with familiarity with the keyboard, continuing through the use of commercial programs of various sorts, and concluding with four weeks of beginning BASIC programming. Off-line topics include hardware (core, ICs, RAM, ROM), applications (business, sports, government), computer-related careers, and social issues. See Appendix C.1 for a complete list.

The other laboratory contains 15 IBM PC Juniors, 12 Apples, and a timeshared Compupro system with five terminals. Except for a few accounting classes, the room is used entirely for programming classes. There are almost always two classes using the lab at once. (The school day has nine 40-minute periods, and there are 17 classes scheduled for this room.) The adjacent classroom is also available to the programming classes, in case the teacher wants to spend the period on a lecture. The lab/lecture decision is often made on the spot, not scheduled in advance. The lab room also has a chalkboard, with some tables and chairs empty of computers near it, so that a teacher can also conduct a mini-lecture during part of a lab period.

The Computer Science Department has five staff members, three full-time and two shared with other departments. One of the full-time members, the head of the department, spends half her time teaching and the rest on administrative tasks; she coordinates computer use for the entire K-12 school district.

Courses Offered

The department offers five courses this year. All are taught during both fall and spring semesters, except for the 6502 Assembly Language course, which is taught only during the spring. The department also cross-lists the Automated Office course, which is offered by the Business Department. Quotations in this section are from the department's official course listing. (See Figure 1.)

Computer Discovery is the introductory course mentioned earlier, which is taught in a separate lab room. "This course is an introduction to computers. It is designed to provide basic terminology and information about computers: what they are, how they work, and how they are used in society. Hands-on experience will be provided on microcomputers. Applications will include: word processing, programming, animation/graphics, telecommunications, and simulations. Topics such as career opportunities and computer ethics will also be covered." This course is being phased out, because the same curriculum is now taught to many students in the junior high schools.

Computer Programming 1 is a beginning BASIC course. "This course is designed to learn how to attack a problem for computer solution and to learn fundamental programming techniques necessary to gain a strong foundation for learning other programming languages." The course has as a prerequisite either Computer Discovery or a semester of algebra. This is the most popular course in the department; nine sections are being taught this semester.

Computer Programming 2 is a more advanced BASIC course. "This course will continue to emphasize methods of designing and writing clear, reliable, and easily modifiable programs in BASIC. The topics covered will focus on working with arrays, sorting techniques, string manipulation, and file concepts. The course assumes a strong working knowledge of good programming practices including top-down design, modular programming, and documentation techniques."

Programming Style is the name of the department's Pascal course. This course is not meant to satisfy the Advanced Placement (A.P.) syllabus of the Educational Testing Service, but they're considering offering a second semester of Pascal which would prepare students

Figure 1. Computer Science course offerings.

for the A.P. exam. “This course will emphasize programming methodology, algorithms, and data structures necessary for the systematic development of software. These skills will be taught concurrently with the computer language, Pascal.”

6502 Assembly Language requires either Programming Style or Computer Programming 2 as a prerequisite. (The 6502 is the microprocessor used in the Apple II computer family.) “After learning the hexadecimal system of arithmetic, students will use a monitor program to examine and alter the contents of memory in order to understand how memory works. The course will then focus on how to use an editor to create source code which is then assembled into machine code and executed.”

Classroom Observations

Before 8:00 a.m., when classes begin, students may use the computers for whatever they like, including playing games. Games are not permitted once school starts. Games are somewhat controversial among the faculty; the department head told me she would prefer not to allow games at all, but another teacher said he supports them. (One reason advanced for opposing games is that many are pirated. Indeed, I observed some disk copying going on in the morning.)

Programming 1 lecture session

I observed a classroom lecture about variables in a Programming 1 class. The class size was about 30, half boys, half girls. There is no textbook; instead, locally-written curriculum units are handed out to the students. The unit about variables is the second in the sequence. Each unit contains explanatory text, examples to type into a computer, and programming assignments to be done independently. (Appendix C.2 shows sample pages from this unit.)

Both the handout and the lecture emphasized the criterion of structured programming. Since BASIC is not a language that provides many tools for program structure, this criterion is manifested in two main forms: an early introduction of GOSUB (the BASIC subroutine call command), and the extensive use of REM (remark) commands. A 41-line

sample program in the variables unit contains 28 lines of remarks and only 13 lines of effective commands. Six of the 13 are either GOSUB or RETURN; the six LET and PRINT commands in the program are divided into three subroutines. (See Figure 2.)

The department's effort to get the idea of modularity across to students is hampered by the lack of local variables in BASIC. The BASIC subroutines are physically separated (by REM lines full of asterisks) but each routine has to know about the operation of each other piece of the program. This state of affairs eliminates much of the real benefit of modularity. In a non-procedural language the programmer cannot, in fact, direct his or her attention to a single subroutine without thinking about the rest of the program at the same time. In the end, the students are induced to program with subroutines largely by the threat of a bad grade if they rebel.

The faculty are aware of the tension between their ideals of program structure and their choice of programming language. They showed a degree of defensiveness on the subject before I asked any questions about it. In the lecture I observed, the teacher told his class that "BASIC has some deficiencies." (The particular deficiency he had in mind was that a BASIC programmer can't use variable names longer than two letters without inviting trouble in the form of reserved words embedded within a name.) The department head, on the other hand, told me that in her opinion, there is no single best language; different kinds of problem are most easily solved in different languages, and "even BASIC" is best for certain problems.

Programming 1 laboratory session

I observed a different Programming 1 class in the lab. They spent the first part of the period sitting in groups around the large tables near the blackboard. Each group worked together to write, on paper, in English, a description of an algorithm to solve a problem presented by the teacher. (The problem was to list all three-digit numbers which are "special," defined to mean that the first digit is smaller than the second, and the second smaller than the third.) Most of the groups chose to present their solutions in the form of a flowchart, a formalism which they had seen in the first few days of the course. The teacher

Figure 2. Structured programming.

Figure 2. Structured programming (continued).

pointed out to the class that they didn't yet know enough BASIC to write a computer program to solve this problem, but that that isn't important; the main goal is to learn how to solve problems, rather than the details of a programming language.

This particular class has two "proctors," students from the more advanced classes who receive course credit for helping out in the beginning classes. The ways in which teachers use the proctors vary greatly. In the absence of a more specific assignment, they wander around the room, helping students who are working on the computers. (During the non-computer activity I just described, one proctor sat with a group which was having trouble and talked through the problem with them.) But proctors are often used as messengers: take this to the office to be copied; ask an administrator to sign this form; go to the cafeteria and pick up my lunch. A different kind of assignment was mentioned to me by the department head, who sometimes has one advanced student read magazine articles about some aspect of computing and make up questions about the articles to be given to the Computer Discovery classes.

Students in the computer laboratory work two to a computer, although each has his or her own floppy disk. They choose their own partners. The atmosphere is generally relaxed. Students talk with one another, both about their assigned tasks and on other topics. I noticed very little tension, even when a defective floppy disk caused the loss of a day's work. (This happened more than once, because the school had recently bought a supply of used diskettes.) Not everyone was equally inspired by the assigned problems, but none of the students I saw seemed driven by fear of any sort. My own presence in the lab was generally accepted without question; several students asked me for help on programming problems.

Programming 2 laboratory session

The Programming 2 class that I observed was working in the laboratory on programming problems involving arrays. The class includes 11 boys and 4 girls. The style of the class was similar to that of the Programming 1 laboratory session, except that the assignments were more advanced. Each problem involved filling an array with numbers and then

processing them in some way. One problem, in particular, was very difficult for most of the students: the task was to go through an array of numbers, copying the odd numbers into a second array and the even numbers into a third. Because this problem requires incrementing two counters at once, it doesn't fit readily into the FOR-NEXT structure as the other problems, with only a single array at a time, could use.

The issue about picking appropriate names for variables surfaced in the same programming problem in a revealing way. The teacher encouraged the students to give the arrays mnemonic names, like N for all the numbers, O for the odd numbers, and E for the even ones. As index variables for the arrays he suggested NC (N Count), OC, and EC. The description of the problem does not say how many numbers will be read; instead, the programmer must read numbers into the array N until a negative number indicates the end of the input. This first part of the problem was easy for most of the students. They wrote a FOR-NEXT loop incrementing NC, and breaking out of the loop with a GOTO when the negative number appears. At this point, a *second* FOR-NEXT loop is needed to divide the array into two smaller arrays. But the variable NC now represents the size of the array N; it is the limiting value for the loop variable, so some other name must be given to the loop variable itself. None of the students whose programs I saw had an appropriate choice; the most common name chosen was X.

Advanced classes

The Programming Style (Pascal) class and the Assembly Language class meet at the same time. I observed a session in the lab, in which the Pascal class was meeting at the blackboard, while half of the Assembly class was working on the Apple computers. (The other half of the Assembly class was in the classroom next door, with their teacher. The students working on computers were unsupervised.) The Pascal class consisted of 12 boys and 4 girls; I saw 7 boys in the Assembly class, and I'm told the others are boys also.

The Pascal teacher was writing a small program at the blackboard. He talked about the idea of top-down structured programming, using terms like "stepwise refinement" which come from Dijkstra's formal treatment of that approach. The problem was to read two

numbers into two variables, then exchange the values of the variables, and then print the results. The teacher divided this task into three parts: input, processing, and output. (This division parallels the structure of the sample BASIC program I mentioned earlier.)

The class agreed that a third variable would be needed to hold one of the values temporarily during the exchange. The teacher asked for names for the three variables, and a student suggested A, B, and C. The teacher considered those names insufficiently descriptive, and proposed instead NUM1, NUM2, and TEMP. The student wasn't convinced that that was an improvement, and laughed at the teacher's insistence about it. "We'll complain, but we'll do it." (Although there was a good deal of argument back and forth throughout this class, the tone was one of mutual liking and respect; I had no sense of a power struggle in progress.)

Computer Discovery

I observed a Computer Discovery class, held in the room with the Pets. There were 11 boys and 7 girls in this session. The class began with a lecture about the day's off-line topic, the steps involved in manufacturing an integrated circuit (IC). The teacher passed around examples of an IC, a transistor, a core memory array, and similar computer-related artifacts. (One artifact was a circuit board containing an IC, which she called "an IC"; another was an IC not mounted on a board, which she called "a chip.") She tried to involve students by eliciting answers to questions like "What is a circuit?" "In what other situations do you use the word circuit?" (The teacher suggested a circuit judge; a student mentioned circuit breakers.) Each student is required to keep a notebook with notes on the class sessions.

The second half of the same period was devoted to the on-line topic of simulations. Students were given a choice of two simulation programs, Lemonade and Market. They were given forms on which to write down and hand in the results of each round of running the program. The programs had been loaded into the computers in advance.

Interviews

I interviewed the department head and one other teacher extensively, and several students briefly.

The difference in tone between the Computer Discovery class and the programming classes reflects differences in the goals behind these classes. The department head mentioned two goals: first, that every student be able to use the computer as a tool, and second, that programming students learn problem-solving skills that will be useful even outside the context of programming.

Okay, when you say “computer-based education” I think I’m going from two places. I’m going from Computer Science Department and from using the computer as a tool. My first goal for all students really would be using [the] computer as a tool, using it for word processing, using it for data base [inquiry]... [to] be aware of what it can do, and when they need it, to be able to go to it and use it. For example, if they need to use the telephone, if they need to communicate with someone, they know whether they need to write a letter, or whether to use the telephone, how they’re going to communicate. And they do it without thinking twice, and I would like the same thing with computers....

My other goal, in the Computer Science Department, is to offer them a wide range of what’s happening in the computer world, from programming to robotics to applications to telecommunications. But I think also [a goal] is to learn how to think through a problem and break down the problem because I think ultimately, even if they never do the problem on a computer, by learning how to think through a problem and to sequence the steps, this will carry them a long way.

She added that she does not think of her first goal as one of job training. “I don’t think that we can, at the high school, truly successfully train a kid for a job.” But she distinguishes preparing a student for a job which is specifically computer-related from preparing him or her to be able to handle a computer, if one happens to become part of any job. The latter *is* part of the goal for all students.

I’m thinking of it in terms of their daily life; I’m thinking in terms of their job; but I’m not thinking in terms of them going out and getting a job to do that. I’m thinking in terms of them being able to go out and in their job, if there’s a computer there, being able to use it, being able to word process, being able to data base.

The other teacher I interviewed, although he describes his goals as still being formulated, presented only one central one: problem solving.

[W]hereas most of the kids, the vast majority of the kids, are not going to be programmers—in fact many of them won’t even be recreational programmers—hopefully there should be some other result of all that time and effort spent... a person who learns programming, I mean learns it proficiently, also learns that library of techniques that they can use in any sort of problem solving, you know, sort of a thought organizer, something that, surprisingly, is not taught... So I’d say basically that I’m primarily looking to teach kids a systematic sort of a problem-solving skill.

He considers this goal “revolutionary” by contrast to the goals of the school in general. “It should have been taught in algebra, but it’s not.” He looks upon the elective status of the programming classes as an advantage in encouraging a flexible problem-solving approach.

Teaching [programming] in an environment of trial and error... with as little [of the] sort of competitive and punitive kinds of circumstances that surround all of the academic courses at school... If you don’t succeed in algebra, you’re dumb. But more importantly, for many people, if you don’t succeed in algebra you can’t get on the right track, go to the right school, and that whole thing. Currently, as an elective, since that sort of feeling doesn’t exist around programming, ostensibly you should be able to approach the process of programming with less anxiety.

He also sees the problem-solving goal reflected in the particular emphasis they use in teaching programming.

The way in which we’re teaching BASIC as a structured problem-solving, top-down, break the problem down into subproblems, deal with the subproblems independently, try to locate your problems and individual subproblems and that sort of technique... of teaching BASIC does go along with this concept of teaching problem solving as a systematic approach to any kind of problem solving.

Computers are so straightforward, you know, and they’re so forgiving. You make an error, you just do it again. Nobody slaps you around for it, and there are no judgments levelled, no morals listed. Yet so many kids seem to reach the wall so easily; it’s surprising to me. It requires a teacher who is, I think anyway, requires a teacher who is kind, who is encouraging, who is sympathetic, and who is persistent... Kids have to be given the attitude that there is no failure here, that sitting down and putting your hands on the keyboard is success. I

mean, for a while, anyway, there's no failure. Maybe later there will be. I mean, I have certain judgments as to what I expect out of kids, but in BASIC 1, if in fact a kid comes on a regular basis and makes an honest stab at attempting to do—I mean, not all kids think in the way computers think. And I don't necessarily see that as a disadvantage. I think all kids can learn to think in a systematic, kind of step-by-step approach, and I think that's important.

Both teachers mentioned that the classroom environment was important to their work. "I think it's a very free, open atmosphere." "I think it's very clear that kids are welcome in that room." But I found it interesting that both of them talked about "the room" in the singular, meaning the programming lab, when making these remarks. My own observation was that the atmosphere was less free in the Computer Discovery room than in the programming lab.

I interviewed half a dozen programming students briefly about their own goals and about their ideas of their teachers' goals for them. They all mentioned the possibility of using computers on the job as one reason for taking the course; in some cases, the student was himself or herself concerned about using computers later, while in other cases, some adult had suggested or insisted upon the course for job-related reasons. The boys I interviewed added that programming is fun; the girls didn't. (I didn't interview enough students to give this split any statistical validity at all, however.) The students were unanimous in describing their teachers' goals in job-related terms. "Learning about computers keeps you from being scared about them, and when we grow up, most jobs will be computer-related, even if they won't be programming jobs." "Because computers are the future." "Because computers are getting so important."

Summary and Conclusions

This school offers a sequence of courses ranging from a literacy course, intended for every student (either here or in the junior high schools), to advanced courses in Pascal and assembly language for expert programmers. The courses form a pyramid rather than a set of tracks: a student is expected to start at the beginning and work up the sequence, but more students start the sequence than finish it.

In the programming courses, the structured programming approach is very much in

evidence. This reflects the goal of problem solving, which was explicitly mentioned by both of the teachers I interviewed. The lowest-level course, Computer Discovery, has a separate goal of awareness and the ability to use the computer as a tool. The goals mentioned in the teacher interviews were consistent with the ones mentioned in the official course listing: “computer awareness, computer applications, and computer languages with an emphasis on programming style.”

The school has made a strong commitment to computer science education. Compared to the other schools I visited, it has a large staff (five teachers) and a large facility (49 computers and terminals), even relative to the school population of 3000. The staff has put much effort and thought into the design of curriculum materials tailored to their own goals. The computer courses serve a large number of students, and a relatively large number of those students go on to advanced programming courses. The atmosphere is that of an institution that knows its job and does it smoothly.

The thing that was most striking to me, in my observations and in the interviews, was the difference in tone between the Computer Discovery class and the programming classes. The same teacher, in these two settings, seemed like two very different people. I think that the contrast is not accidental; it reflects a real antagonism between two sets of goals.

The Computer Discovery class is based on the “literacy” metaphor: the idea that every student needs an “exposure” to computers, a sort of inoculation against computer phobia, for pragmatic reasons in later life. Since not every student is equally interested in computers, and not every student is equally well prepared for the challenge of original problem solving, this metaphor leads inexorably to an intellectually weak curriculum, full of arbitrarily chosen facts to be memorized. In the lesson I observed, some of the facts were technically obsolete (core memory), while others were downright wrong (“chip” and “integrated circuit” presented as two separate ideas). The hands-on part of the course is better, but the tone of a simulation session is changed to just another jump through a hoop by requiring the students to fill out forms documenting their program runs.

By contrast, in the programming classes the assumption is that students must feel free and at ease in order to display creative problem solving. Most students choose the programming classes out of a real desire to learn the subject, not as a lesser evil. Teachers

are able to fool around with the kids. Ideas, not facts, are the main stock in trade.

The contrast is not surprising. The important thing, though, is that the teachers don't seem to think of the contrast as a contradiction. That is, the differences in style would be sensible—if politically incorrect—if the department's model were that smart kids take programming classes and other kids take Computer Discovery. But in fact they think of these courses as a *sequence*; Computer Discovery is a prerequisite for programming. So they teach two very different messages to the same kids, and I believe that in the Computer Discovery class they are working against their own goals for the programming classes. The lesson of the programming classes is that "there is no failure here," that there is more than one way to solve a problem, and that computers are forgiving and flexible; the lesson of Computer Discovery is that learning about computers means, at least in large part, learning technical jargon and arbitrary facts.

On the other hand, I think the aura of smooth success in the computer center is a result of the unusual degree to which the department *has* goals in the first place, and takes pains to make its teaching follow the goals.

School B

Description of the Facility

This is a school of about 1800 students in a middle-class town. The population is largely white, but with a significant Chicano minority. Computer courses are offered by the Science department in a combination lab and classroom.

The room has standard classroom seating facing the blackboard in front; the rear half of the room has 10 Apple IIe computers. Each is equipped with both a color and a monochrome monitor. (The latter permits higher resolution text: 80 columns instead of 40.) Two more Apples are at the front of the room. One of these is on a wheeled cart, and is often borrowed by teachers in non-computer courses. The other is dedicated to the operation of an electronic bulletin board system maintained by the teachers and students. This latter machine has a hard disk drive (with a larger capacity than the usual floppy disks) attached. It is this electronic bulletin board that makes the school's computer facility unusual.

This laboratory is dedicated to classes specifically about computers and computer science. The school has begun to install computers elsewhere for use in other areas; for example, computers are used in business courses. The district has plans to expand the use of computers in other fields, but has made the computer science lab its first priority and is just beginning the other areas.

Courses in this lab are taught by two science teachers, one of whom is the department head. Both of them also teach non-computer courses.

Courses Offered

There is only one formal Computer Science course in the school. It is a year-long course, and five sections are taught. Each section has about 25 students. The first semester is about computer applications, primarily word processing and a data base program. A little Logo programming is also included in that semester. The second semester is about BASIC

programming. Some students take only the first semester.

Students with special curriculum requirements are enrolled in the same course but given individual instruction. This category includes a few advanced students studying Pascal and also a few who want a second semester of working with application programs.

Classroom Observations

The machine that runs the electronic bulletin board has the place of honor at the front of the room. Although the bulletin board is not involved in the computer course curriculum, and only a few of the students are actively involved with it, it is clearly the pride of the department.

The side and rear walls are covered with computer printouts and with posters announcing computer-related events. The events include ones addressed to teachers, like regional conferences about educational software. The computer printouts are in categories. In addition to the expected BASIC programs and Logo programs, there is a category called Zork containing transcripts of interactions with this role-playing adventure game. A fourth category is a menu of the types of information available through CompuServe, a commercial timesharing service to which the school subscribes.

Ordinarily, during a class period, half the students are in the front of the room hearing a lecture while the other half are using the computers in the rear. I visited near the end of a quarter, on a "catch-up" day; students using the computers were working on old assignments or playing games. There were also students in the seats in front, either doing homework for other courses or talking with one another; these students were either finished with all their assigned work or so far behind that they had given up. All of these various activities were allowed by the teachers. The classes were fairly evenly divided between male and female, although on this special day most of the boys were actually using the computers (some to work, some to play games), while most of the girls were in the front.

The teachers show some ambivalence about the playing of computer games. In principle, game-playing is an approved activity. Game transcripts are posted on the wall, and teachers take an interest in the latest games their students bring in. Even the illegal copying

of game software is officially tolerated and openly practiced. On the other hand, students are usually allowed to use the facility during the lunch hour for serious work, but not for games. (Officially, the computer center is closed during lunch hour, but usually one of the teachers stays in the room or else a trusted student is placed in charge. But the door is locked, and students need a good reason to be admitted.) On the day I visited, the department head had just decided to ban video games altogether because of damage to joysticks and keyboards from repeated hard use in game-playing. But the rule had not been announced publicly and was only partially enforced.

Students are graded on the basis of assigned projects. At the time of my visit, most of the students were working on a recent assignment, a program to plot graphs of trigonometric functions on the screen. Points are assigned not only for the correct working of the program but also for “creativity” and clarity of documentation within the program. For example, drawing axes on the graph was not a required part of the assignment, but was considered a valuable addition to the minimal program. Students are encouraged to use one another as sources of information and help.

The electronic bulletin board

When someone calls up the electronic bulletin board, it is possible to watch what that person is doing on the screen of the Apple in the front of the room. Most of the use of the bulletin board is outside of school hours, but someone did call in during my visit, and several people wandered over to see who it was. It turned out to be a regular user, a high school student from another city in the area. He was identified to me as a “hacker,” meaning that he enjoys knowing how the program works and looking for ways to make it fail. There is some ambivalence about that, too: the department head talks with sharp antagonism about “hackers” in general, but has a friendly relationship with this particular one.

An electronic bulletin board is a collection of public messages. The messages are arranged into categories by topic: computer programming, politics, music, and so on. (Appendix C.3 shows sample messages from this school’s board.) At this school, each

category is supervised by a particular teacher or student. The task of a supervisor is to stimulate discussion, to remove messages after some length of time, and to police the messages, removing any that are libelous or otherwise in violation of the rules of the board. Anyone who calls up the computer can read the messages; permission to post new messages is assigned by the supervisor for each area. In this particular bulletin board, the areas are named after physical locations in the school, such as the cafeteria and the library. As in an adventure game, the program uses the metaphor of a map, and the user moves through metaphoric space to get from one area to another. (Appendix C.4 shows the map, and Appendix C.5 lists some of the commands used to move through the messages.) Although programs to run a bulletin board are commercially available, the school has chosen to write its own software, a task started by the department head and continued by him and by students.

During one of the periods when there was no class in the center, the department head arranged to have one of his advanced students demonstrate for me the use of the computer and modem to call up another bulletin board. (She was excited about the prospect because she got to miss a history class for the purpose.) There is only one machine with a telephone connection, so the school's own bulletin board is off the air while they use the machine to make outgoing calls. In this case, the call was to a bulletin board run by a high school in another state. That second board also provides an electronic mail service (that is, messages can be addressed to a particular person as well as to the general public), and we started by reading some mail addressed to the student who placed the call. The mail was from students at the out-of-state school, with whom the student here had become acquainted only through this electronic communication. The mail wasn't about anything special, just anecdotes from daily life. As it turned out, we never did read any of the public messages, because one of the students at the other school started a conversation with us. (The conversation was typed, not by voice, but it was immediate and interactive, not like mail.) A few other students joined in at the far end, and the department head and I suggested things to say at the near end. There was no teacher present at the other school's computer center that day. ("Our sysop [system operator] isn't here today, so we're running wild," we were told.) Much of this conversation followed a common pattern for such electronic dialogue: making

up stories about oneself. Without voice or visual clues, anyone can take on any personality, limited only by the imagination (and skill in verisimilitude). Our correspondent used a pseudonym and an assumed age for the first part of the conversation, before telling us who he really is.

Communication between the two schools isn't all gossip. Programs are exchanged electronically, along with ideas about new projects. The telecommunication facility also makes it possible to use CompuServe, a commercial service that provides access to several large data bases as well as bulletin boards and electronic mail. The "real" data bases on CompuServe fit in well with the use of a data base program locally, on the Apples, which is part of the computer curriculum.

The use of telecommunications by students is limited by the cost of long distance telephone calls and of CompuServe access fees. Since a relatively small number of students actually use this capability, it's hard to justify spending a lot of money on it. One of the preoccupations of the department head is the search for creative funding sources. Some money is available through Gifted and Talented Education grants.

Interviews

At this school, several levels of goals are in evidence. When I asked the department head about educational goals, he immediately replied with a question, "Mine or the district's?"

The school district that includes this high school set up a committee several years ago to establish a K-12 policy statement for computer education. The statement includes a "district philosophy," a list of seven goals, and a detailed chart of curriculum topics for the various grade levels. See Figure 3 for the philosophy and list of goals. Three of the seven goals are about staff training and adult education. The four goals directed toward students are to teach "the capabilities and limitations of computers," "the impact of computers on society," "skill in the operation of basic microcomputers," and "skills necessary for appropriate use of computers in a vocational or college situation."

These official goals do not include the ability to program a computer. The specific

Figure 3. Philosophy and goals.

curriculum topics do include “write a simple program involving a decision” at the junior high school level and “programming languages overview” at the high school level. There are also topics *about* particular kinds of programming, including “robotics” and “algorithms” in junior high school and “artificial intelligence” in high school. (Figure 4 shows the complete table.) Not all of these topics have actually been implemented in the curriculum as currently taught, so it’s too soon to know the extent to which they’ll involve actual programming experience.

However, Logo programming is introduced in the early elementary grades and continued through high school in this curriculum plan. The use of procedures starts in grades 4–6. The use of Logo is seen not as real training in programming skills, but rather as a way of showing students what programming is like without actually doing any. (This view of Logo is reflected in the actual programs posted on the wall in the high school computer center. Students used several procedures in drawing a complex picture, but none used procedures with variables, or any control structure more sophisticated than straight-line sequences of commands.)

The second layer of goals is that of the explicitly expressed goals of the individual teachers. Here is the department head:

My personal goal in teaching the computer class is to have the student become involved enough with computers, at the end of the class, to know whether or not they want to proceed with that involvement, and to be aware enough of the basic applications like word processing that if they did want to proceed they could.

This is a remarkable goal statement, because it sounds like a retreat. The district wants students to learn how to use a text editor. This teacher wants them to learn whether or not they *want* to learn how to use a text editor! But this is not the entire story; more goals became apparent later in the conversation.

The other teacher I interviewed began by reiterating the district’s emphasis on familiarization and preparation for jobs. “At the high school level you’re not really trying to turn out professional programmers, you’re just trying to get people used to seeing machines because obviously a lot of the jobs that they’re going to be doing years from now are going to be using computers.” But he also has an interest in teaching structured programming

Figure 4. Curriculum topics.

languages.

We do, of course, have the college-bound kids in mind, too.... In particular, Pascal seems to be the language of choice throughout the colleges these days, so for those students that can handle it, we'd like to introduce them to Pascal.

[P]rogramming a computer takes a kind of a train of thought. You can be very disorganized about it and write a BASIC program, we say in linear style; you start from step one and just sort of sit and write. On the other hand we try to get them to write in a structured way. That's one of the nice things about the language Logo.... In Logo you write procedures. This is one of the reasons we give them Logo before we give them BASIC, so that they get into the habit of writing procedures for individual parts of the program...

Another, deeper layer includes goals that are not articulated in response to a general question about goals, but are strongly articulated in response to specific questions. I was struck by the number of examples of advanced students, both in my own observation and mentioned in the interviews, who are female. I asked the department head about it.

Ten years ago, there was a very serious problem in this district of girls not taking science and math. The problem centered on female counselors, two of whom have since retired and left the district. My wife, who went through this district, encountered the same thing. So I have kind of a special sensitivity to that. Whenever the science courses, and I'm science department chair, do not show a balance I go through a real thing. But I haven't had to go through that thing very much because everybody knows that they're not going to go telling girls they're gonna take sewing and typing and all this kind of stuff. And we do have in fact in the physics and in the computer courses a slight majority of girls. Not because it's word processing, either. There is a word processing course in the school, in the business department. There is an ROP [Regional Occupation Program] word processing; there is an ROP data processing. I don't know what their percentages are. But there is an awareness at this school—I'm not saying this to B.S. you; it's taken ten years—that girls will take these courses. And they don't shy away from it. Some of the best students... in fact, most of them working on the bulletin board are female.

This is a clear example of a strongly-held goal, well reflected in the school's practice, and yet not explicitly considered as one of the official goals.

What about the goals behind the computer bulletin board system? Although "data

bank access” is one of the topics in the official curriculum, for a school to run its own bulletin board is quite unusual, even though many schools espouse “computer literacy” goals very much like the ones espoused here.

Well, I do that for three reasons. Number one, there are students who are very talented, and either we don’t know what to do with them or we provide some kind of structured situation that we can deal with while teaching a general class, and yet they have the ability to really do something, the phrase in gifted and talented education is “qualitatively different.” Often these students are identified gifted students. We have three or four students here who have run their own bulletin boards at home, who know BASIC programming, who know machine language for some system.... If they take a regular class, they’re not going to be dealt with very successfully. And if we just let them loose, they’re going to go off and start running off illegal copies of games, and start hacking away at this, that, and the other, and inadvertently damaging equipment. So it provides a structure for them.

The second reason is that I believe telecommunications is probably going to become the second biggest arena of use of computers over the next five or ten years. And I would like to have at least a core of the most talented students here be aware of that, be able to use it. They’re probably the students who are going to go to a college or university and become involved in accessing libraries via computer, accessing data bases via computer. They’re the people who may become hired by AT&T and MCI and running those systems. We’ve already had some students hired while they’re still in college, and one while still in high school, in those industries. So that’s why we went that route.

We also went that route because it’s not a very difficult one to get into. We chose the most difficult way; we wrote our own program, we created our own system. But for \$100 you can go out and buy a program and you can run your own system off of two disk drives. Any school can do that. There’s only five in the country doing that right now. We’re kind of hoping to encourage others to get involved. It’s a touchy area, too, because people like to post things that are not legal, and that becomes an ethical thing. And we really have to work with some students who really want to post Master Charge numbers and phreaking numbers. We absolutely prohibit that. We do not, however, prohibit students posting information relating to how to crack programs and work with disks. I really view that as a learning experience, and I will stand that ground against whoever says otherwise. I take my guide from [Apple Computer founder Steve] Wozniak, whom I’ve seen in a couple of meetings. He made it very clear that

he learned his expertise by hacking, and I don't think there's anything inappropriate in a student learning how to... learning machine language and... I don't know, I wandered a lot in answering that.

(By the way, Wozniak does not in fact make this distinction between copying software and fraudulent use of telephone service. He thinks even the latter is okay, when done by young people in an exploratory way. He was quoted in the newspapers in February, 1985, in support of a high school student threatened with expulsion for changing his grades in his school's computer.)

Summary and Conclusions

This school tries to meet the computing needs of all students through a single year-long course. The first semester of this course teaches computer applications; the second semester teaches BASIC programming. Students with special needs beyond that curriculum are accommodated through independent study arrangements. The school district has prepared a K-12 curriculum plan that is more ambitious than this, but it has not yet been implemented fully.

The school's official policy statement on computer education embraces the computer literacy metaphor and cites the use of computers in the economy as the main justification for computer education; the course is not specifically vocational in the sense of training for particular jobs, but job-related skills like using a data base are emphasized. In particular, the department head mentions the telecommunications job market as an important justification for the expense of running an electronic bulletin board. Another teacher mentioned structured programming and problem solving skills as goals, but only for a small number of students in independent study programs.

I want to peel another layer off the goal onion. It seems to me that an important reason for the electronic bulletin board at this school is that the department head, himself, wants to run a bulletin board. If he weren't a teacher, he would run one in his home. Since he is a teacher, he combines his hobby with his job. That's why, after summoning a student to demonstrate communication with another school, he ended up telling her what to type most of the time!

I see nothing wrong with that. On the contrary, I think that a teacher's own *personal* goals can be a very deep source of learning for his or her students. I point it out because people tend not to talk about such goals; it is considered more dignified to have abstract goals, applicable to an entire student body. I consider it an example of this tendency that the department head talks about disliking hackers, although in fact he is one himself! He takes pleasure, for example, in the task of "bulletproofing" the bulletin board program so that it won't fail no matter what its users type in. He does not see this bulletproofing as a chore imposed by the destructiveness of "hacker" users; he enjoys the challenge.

The combination of lackluster official goals with vibrant, strongly held personal goals should produce, in practice, a laboratory in which the interstitial activities are more interesting than the official curriculum. Indeed, that is just what I observed. The telecommunications focus is the best example; the interstate conversation I witnessed was clearly a thrill for all the participants. But it is not the only example; I was told more about the independent study students—both the ones learning Pascal and the ones learning an extra semester of data base manipulation instead of programming—than about the mainstream students.

I believe that this computer facility could be even better than it is if its unofficial goals held more weight in official decision-making. For example, the high cost of long distance telephony restricts the kind of interstate conversation I saw to a small number of students. There is only one machine equipped with a telephone interface, so only one conversation can take place at a time. To initiate an outgoing call requires turning off the bulletin board service. If this school had a timeshared minicomputer instead of a separate Apple for each student, the same sort of computer-mediated communication could take place within the school, for many students at once. The bulletin board could be used from several terminals scattered around the school, instead of being limited to those students who have computers and modems at home. The entire student body could send and receive electronic mail. Large data bases, like the ones at CompuServe, could also be maintained locally at no extra cost. The cost per workstation would be higher than it is for separate microcomputers, but the equipment would be used more effectively, because it would be better matched with the actual teaching goals of the department.

I would like to mention again the school's remarkable success in meeting the goal of sex equity in computer education. In most schools, girls are well represented among the beginning students, but not among the more advanced group. That is not the case here; the faculty has pursued the goal of sex equity and achieved it.

School C

Description of the Facility

This is a school of 1800 students in a well-to-do Silicon Valley suburb. The computer laboratory is used for programming courses offered by the Mathematics department.

The facility consists of three connected rooms. One is a classroom containing 10 Hewlett-Packard microcomputers. The second is a classroom containing 15 Apple IIe computers connected to a hard disk drive through a Corvus network. The third is the office of the facility manager, who is a teaching aide. It contains a few standalone Apple IIe systems. The classrooms have blackboards, but students sit in front of the computers even while listening to a lecture.

Generally, at most one of the two classrooms is used for formal classes in any period. Therefore, at least one of the rooms is available for informal student use all day, except during one period while the lab manager is at lunch. (Her lunch period is not during the student lunch hour, a popular time for informal computer use.) There is a “usage log” in each classroom, a notebook in which students are supposed to record their non-class use of the computers, and the purpose for which they use them.

In addition to the three programming courses taught by the Mathematics department, the facility is sometimes used for word processing by typing classes. There are also computers elsewhere in the school; each department has an Apple IIe, and the Mathematics department has four IBM PCs. The computer lab is also used by a computer club that meets once a week during lunch hour. (The club has seven or eight “core” members and another 20 or so who show up sometimes.) Every freshman is introduced to the facility through a one-time “exposure” session.

On the walls are news clippings and program listings under the headings Projects, Jobs, Contests, Ethics, Applewriter, Pascal, Applesoft (a version of BASIC), and Media. The word “ethics” is in big letters on both front and back walls.

Courses Offered

Three mathematics teachers teach one programming course each. There is only one section of each course per semester.

There is a beginning and an advanced Pascal course. The latter is taught by the department head. The beginning course covers the first eight chapters of the text *Oh, Pascal*; the advanced course completes the book. The Pascal courses are taught using the networked Apples.

The third course is called Computer Math; it teaches BASIC programming using mathematical examples. It is taught using the HP computers. It has Algebra II or Trigonometry as a prerequisite, although this rule is not strictly enforced.

The BASIC course is *not* a prerequisite for the Pascal sequence. Some students learn both languages, but some choose one or the other.

The aide who manages the facility also offers an independent study course called Computer Literacy for a small number of students (two this semester, five last semester). This course is not part of the mathematics department. It has no fixed curriculum; each student negotiates a topic with the teacher.

Classroom Observations

When I arrived first period, there was no class in session in the computer center. Two students dropped in to use the Apples. Since these machines are on a network, they do not have floppy disk drives. Therefore, students can't bring in their own software to use on these machines. That eliminates the issue of video games and of illegal copying of software on these machines. (The HP machines and the smaller number of Apples in the manager's office do have disk drives.) The software available on the network includes educational games to teach, for example, physical laws. Drop-in students can play these games, which include animated graphics games as well as CAI-style quiz programs.

I was introduced to one of the independent study Computer Literacy students. He was studying Artificial Intelligence. The school does not have the hardware or software to support real AI programming, so all he could do at school is collect articles on the subject

and read them, but he has also spent some time at the Heuristic Programming Project at nearby Stanford University.

Computer Math

The Computer Math course began in a separate classroom in another building. (The school has a large motel-style campus.) The teacher assigned programming problems from the textbook and explained how to do some of them. The course is graded on the basis of the number and complexity of the problems each student completes. That is, in this assignment, three relatively simple problems are worth a C. Two additional problems earn a B, and two more beyond that get an A. The C problems were all about generating tables of functions; one was about possible areas of a rectangle with a fixed perimeter; one was Pythagorean triples; the third was the factorial function. The B problems were to verify empirically the formula for the sum of the first N integers and to chart simple versus compound interest. The A problems were to approximate the limit of $(\sin x)/x$ as x approaches zero and to balance a checkbook. In discussing some of these problems, the teacher started writing a BASIC program at the blackboard, soliciting suggestions from students, then left them to complete the program on their own. For the last 15 minutes of the class, we returned to the computer center, where the students worked on the HP computers. There are more students than computers, so some worked with pencil and paper while waiting for their turn. (Most class sessions meet in the lab for the entire period, and students who do not get a computer at the beginning of class take over for the second half of the period.) Twelve of the 16 students I saw are boys; all are seniors except for two juniors and one sophomore. (Students are supposed to have taken Trigonometry before this course.)

Beginning Pascal

The beginning Pascal course started with students working on a problem that had already been assigned. The teacher offered to discuss the problem at the blackboard, but said that anyone who didn't feel a need for that discussion could continue to work separately.

Most students chose to take part in the discussion, but several did work on their own. The discussion took only part of the class time; after that, everyone worked on the computers. Some of the students worked in pairs, because there are slightly more students (17) than computers (15). A majority (10) of the students are girls; all four grades are represented. I was told that at the beginning of the semester there were more freshman boys in the class, but several dropped the course because it required more work than they had expected.

Advanced Pascal

The advanced Pascal course is being offered for the first time this semester, and the teacher is learning the material along with the students. He had picked out a problem from the textbook that he didn't know how to solve, and I was drafted to teach the class about it. What made it a hard problem was not some detail about Pascal but rather the fact that the problem calls for a recursive solution, and he was looking for an iterative method. I explained that at the blackboard, and then everyone started trying to turn my explanation into a Pascal program. (There wasn't enough time to finish the program that day, so I don't know how successful they finally were.) The students seemed perfectly comfortable with the idea that their teacher didn't know the answer either, and brave about trying the problem themselves.

Informal use

During lunch hour several students came in to use the equipment. One student had a diskette of Robot Odyssey, an educational game, that he wanted to use on one of the Apples in the manager's office. She was reluctant to allow him to use it because in general they do not permit the use of "commercial disks"; this policy is partly based on wanting to reserve the facility for educational use, rather than playing video games, and partly to avoid problems of software piracy. But since this student had a legal copy of the program, not a pirated copy, and because the program is a well-respected educational game, she allowed him to use it. When a second student wanted to load the same disk on a second machine, however, she would not allow that, on the grounds that commercial software is licensed for

use on one machine at a time. (The school is careful to obtain multiple-use licenses for the programming languages it uses; they do not have any commercial game or educational software on their networked Apples.)

I asked whether the use of networked computers with a shared file system led to the other kind of computer ethics problem, that of people breaking into other people's files. But individual students do not have private accounts, because the Corvus system preallocates a fixed amount of disk space for each account and that limits the number of accounts that can fit. Instead there are class accounts and a few public accounts for drop-in students. This arrangement seems to eliminate the motivation to crack the password system.

Interviews

During my conversation with the Artificial Intelligence student, he and the facility manager told me about a group of kids who "hang out" in the computer center and are recognized as one of the social cliques of the school; the other students call them "computer nerds." (This particular student is not one of that group, despite his strong interest in computers.) Every school seems to have a few students who are particularly expert about the computers, but this was the first time that those students were called to my attention as a *social* group.

My impression of a "computer center family" was strengthened by the fact that the manager of the room was able to introduce me to several students, describing not only their personalities and their special areas of computer expertise but also some details of their lives outside the computer center. With that background, I expected to see more drop-in use than I actually did see, especially during the lunch hour. But those students who did show up at lunch did seem to feel at home, rather than like clients at the library.

The facility manager told me that she sees a need for a real, hands-on computer literacy course that could be offered to a large number of students. She is not eligible to teach such a course herself because she is classed as an aide; she is not a certified teacher. A recent survey showed that 70% of the current sophomore class have home computers; this means that most of the student population already knows what a "literacy" class would teach,

but she sees it as an equity issue for the other 30%. She is also concerned with sex equity, and is looking for software that would be more attractive to girls.

By contrast, the head of the Mathematics department sees no need to teach students how to use a computer or how to use applications programs.

[My goals are] strictly to teach programming and a language. I figure that the kids get on and figure out how to turn it on, then they can figure out how to use any software at all... I enjoy programming. And I think that people who get into computers should be able to program. And I have seen very little [application] software that I thought was very worthwhile.

He shows no interest in job training, and doesn't talk about computers taking over the world. For him, programming is an intellectual discipline; his answer to "why do you teach programming" is fundamentally no different from the answer to "why do you teach geometry?"

I think that probably if it were starting today I would probably not be involved with it. Because I was a self-taught computer programmer, about 20 years ago, and it was very interesting. But today there's enough in the math curriculum that has to be done and I probably would have stayed with the pure mathematics, and let the computer lab run through the computer lab itself.

He thinks that students take computer courses mostly for fun, rather than as preparation for a job. This opinion was supported, although not overwhelmingly so, by my interviews with students. Beginning Pascal students said "I needed an elective and I wanted something I might be interested in"; "I figure everything you do later in the world will have something to do with computers"; "For both my own personal knowledge and also for work"; "For fun." Advanced Pascal students said "The first reason is I needed to make up credits... and I found out that I'm enjoying it"; "Because I'm interested in computers"; "I've been working with computers for two years and I was interested in learning Pascal." The Computer Math (BASIC) students were somewhat more pragmatic: "I felt that I would need some computer skills"; "I think I need to know something about computers before I go to college"; "Just to have the experience"; "I have some interest in computers, and I thought it might be useful." When I asked a student specifically whether he or she was taking the course for fun or for job training, almost all said "both."

It's quite unusual for a school to offer both BASIC and Pascal classes but not to think of them as a sequence, with BASIC a prerequisite for Pascal. I asked about the rationale behind this policy.

We always taught through the math department a course called Computer Math. And as we originally set it up, it was to teach BASIC with math applications. And we became very strong in the math applications.... When we got these new [Apple] computers and we had the choice of another language I had to find some materials to teach this new language and I chose the book *Oh, Pascal*, which is not math oriented as much as programming oriented. And I liked it very much... I did not even offer a computer math course. And so it was only this year we decided we should go back and have the other also. Especially getting the computers from Hewlett-Packard. Those were not really good for Pascal.... [A] teacher was interested in doing it, too. That has a lot to do with it. I think it's sort of a haphazard thing. The reason that we teach Pascal here is that I was tired of playing with BASIC.

Some of the students look upon BASIC and Pascal as a sequence, while others don't. (Since the BASIC course is made up mostly of seniors, very few can in fact go on to study Pascal while still in high school.) Some of the BASIC students said "I thought I needed to know something about BASIC before Pascal" and "I thought BASIC would be easier to pick up"; others had less logical reasons for their choice: "I didn't know we had a Pascal class" and "The Pascal class was full, and it's a different period." Similarly, some of the beginning Pascal students told me they had already learned BASIC at home, while others chose Pascal for different reasons: "I think you have to have a lot of math for the Computer Math class"; "I just signed up for computer programming."

I asked the Mathematics head about the emphasis on ethics in the computer lab.

When we were tied to the 3000 [a Hewlett-Packard timesharing system], we had all of the school records on them and everything else, and security became a big thing and every year we would have at least one kid who cracked security. I mean, this is the way you play the game. So the ethics in those days was don't try to crash the security scheme, stay in your own account. Lately there's been some copying of software. And it's not just in the kids, but it's with the adults here too, who see a good program and come in and transfer it off. So the emphasis now is on the ethics of copyright laws as well as the ethics of staying within your account.

I asked whether the ethical issues were part of the curriculum for discussion in programming classes.

No. We don't. I mean, the young man who was the lab assistant last year that ran this got a message because he was worried about the amount of copying that was going on. And so he put up the ethics signs. But then when he left at the end of the year, all that stuff just stayed up. But the teachers in the lab, we never made a point of it.

Summary and Conclusions

This school offers a relatively small selection of computer courses: one section of each of three programming classes. There is no course designed to teach non-programming computer use. The two Pascal courses form a sequence, but the BASIC course is neither a prerequisite to that sequence nor a lower track. In fact, it has a mathematics prerequisite that the Pascal sequence does not have. Some students study both languages, while others pick one of them for a variety of reasons.

The use of the computer facility by students outside of the formal class structure was not very great in any of the five schools in this study, but was higher at this school than at most. Such informal use is supported by the provision of adult supervision by a person who is not tied to particular courses.

Policy decisions in the computer facility do not seem to be based on any central administrative plan. Instead, each of the staff members actually involved sets the goals for his or her own work. Of the two I interviewed, the facility manager thinks primarily in terms of computer literacy, while the Mathematics department head thinks primarily in terms of problem solving.

I found many excellences in the computer education facility at this school. Its course offerings are brave and intellectually rigorous. The Pascal classes use an excellent college-level text. The BASIC class wisely concentrates attention on the mathematical problems rather than on praising the defects of the programming language. The computer center encourages informal use and has built up a community of enthusiasts. I felt more at home here than in any of the other schools I visited.

On the other hand, the facility is underused. With three rooms and a full-time man-

ager with no formal teaching obligations, the computer center should be bustling all day. Students should be dropping in to type their English papers on the computers. There should be more sections of the programming courses.

One reason for this contradiction, suggested to me by one of the teachers, is the nature of the student population. In this affluent town full of computer professionals, many students have computers at home more powerful than the Apples at school. Some think they already know everything the school can teach them about computers. This is unfortunate, since a powerful computer does not necessarily provide a powerful intellectual experience; it depends what you do with it.

But I think another reason is that the computer facility is a two-headed beast, with the two heads pulling in different directions. Intellectual leadership is in the hands of the Mathematics department head, who sees computer programming as a mathematical discipline rather than as a practical necessity. This is a strong, defensible position; it is the high school version of what Seymour Papert says about the use of Logo with younger children. But he has other demands on his time, and isn't interested enough in programming to take on apprentices. The practical leadership, therefore, is in the hands of the teaching aide who manages the room. She has quite a different vision, in which the computer is a practical tool. That, too, is a defensible position; it is much more like what I have heard at other schools. She has the energy and the enthusiasm to develop a community of computer users, but she doesn't have the academic machinery to support it, nor the expertise to push it into excellence. The striking difference between these two leaders in their positions on the importance of teaching about ethics is another example of their conflicting views.

The school is planning to hire three new math teachers next year, all of whom will be competent Pascal programmers. Perhaps this added staff will provide a push in a consistent direction.

School D

Description of the Facility

This integrated urban high school serves about 2300 students. The computer facility is under the Business department; its equipment is funded by vocational education grants.

The facility consists of two adjacent rooms. One is a traditional classroom, except for the three hardcopy terminals in the back. These terminals are connected by telephone lines to a timeshared mainframe computer in the school district office downtown. The other room is the computer lab; it has 30 Commodore 8032 microcomputers. The computers are wired together in groups so that several computers can share a single floppy disk drive. The walls of both rooms are filled with student-made collages made from advertisements for computer equipment and software.

The facility is used by two full-time teachers. Each teaches five sections. They switch rooms about every two weeks, so that each class gets alternating hands-on units and lecture units.

Courses Offered

The department offers two computer courses. The first is called "Introduction to Computers." (An official course description is in Figure 5.) Seven sections of this course are taught. Although the course does include some BASIC programming, most of the hands-on work uses application software such as word processing and spreadsheets. The lecture part of the course is mostly about data processing concepts and terminology.

The second course is called "Computer Programming." (Its description is in Figure 6.) It is a continuation of the first course, using the same textbook, but the hands-on work consists of more advanced BASIC programming rather than using commercial applications.

The textbook, *Introduction to Computers and Data Processing*, is lavishly illustrated with color pictures, mostly of large mainframe computers. The text covers the early history of computers and some of the ways in which computers are used in data processing. The

Figure 5. Introduction to Computers.

Figure 6. Computer Programming.

technology described in the book is oriented toward magnetic tape equipment; terminology like “interblock gap” and “sequential recording” is taught. Other chapters deal with the organization of data bases. The only actual computer programming in the book is a short appendix containing an introduction to BASIC. A student workbook accompanying the text provides review tests (Appendix C.6) and programming problems (Appendix C.7). The programming problems are business-oriented, designed to illustrate some of the concepts discussed in the main text.

Both courses are year-long; it is a district policy that all high school courses should be year-long to simplify scheduling. Each section is limited to 30 students, so each student can work individually on a computer. The official course descriptions were written by an administrator at the district office, in accordance with the district policy to create a uniform curriculum for all the high schools in the city. (There has been some criticism of the district because this particular school is seen as better than others serving poorer and more heavily black parts of the city.) The descriptions were written after the courses were already being taught, but they reflect the administrator’s view of what the courses should be, rather than the teachers’ views. For example, the introductory course actually includes more BASIC programming than the district office would prefer.

Classroom Observations

The computer lab is open for about an hour every morning before the first class begins at 8:25. Students can use the computers to work on programming assignments, as word processors (often for homework in other classes), or to explore programming on their own. Game-playing was allowed in the past, but is not allowed this year. Several students enjoy writing programs using the character graphics capabilities of the Commodore computers; graphics projects have a very different flavor from the business-oriented programs in the curriculum.

In a given week, one of the two teachers uses the computer lab every period, while the other teacher is in the classroom every period. Therefore, I observed only the lab work of one teacher and only the classroom work of the other. The teacher I observed in the lab

teaches two sections of Introduction and three sections of Programming; the one I observed in the classroom teaches five sections of Introduction.

Laboratory sessions

I saw both an Introduction class and a Programming class in the computer laboratory. The style of the two classes was similar, except that the Programming class was working on more advanced programs. (Earlier in the year, there would have been a more substantial difference, because the Introduction class wouldn't have been programming at all.)

The teacher began the Programming class by explaining one of the assigned programs at the blackboard. He asked students to supply details about the problem's requirements, and he used this information to draw a flowchart. Flowcharts are used extensively in these classes, as a way to focus students' attention on the overall program structure before they begin coding. During this explanation, some of the students wanted to begin working on the computer instead of paying attention; the teacher solved this problem by threatening to turn off the power to the machines. After the flowchart was finished, the teacher told the students to begin work on their programs. Not all students were working on the same problem; some take more time than others to finish a program.

The Programming class had 15 students, 13 boys and two girls. They were evenly divided between white and black, with one Asian. The Introduction class I observed was larger, with 13 boys and eight girls. This class contained ten blacks, seven whites, and four Asians.

A sign on the wall above the blackboard lists "required sections for programming problem solutions":

1. identification section (student's name, problem number, etc.)
2. variable list (REM statements identifying the variables used)
3. DATA statements
4. processing statements

This way of structuring a program is reminiscent of COBOL, although the sections are not exactly the same as COBOL's divisions.

The teacher spends most of his time, in the laboratory, walking around the room

answering questions from students. Most of the Programming students were coping with the problems; some of the Introduction students, though, needed help with the simplest steps of writing a program.

On this particular day, the lab teacher spent some time working on a computer himself, because he was having trouble solving one of the programming problems (Figure 7). Although solutions are available in a teacher's guide, he prefers to work out each program himself. The difficulty in this problem seems to come from the need to make two equality tests (salesperson and book title) at once. A few students solved the problem before the teacher did. This didn't worry him, but he was apologetic about the lack of modular structure in the solution he did eventually reach.

In both classes, students consulted with one another as well as with the teacher. Talking and moving around the room are permitted. Some friendly conversation took place about topics other than programming, but the students all seemed to be working as well.

Classroom session

I also observed a session of the Introduction course meeting in the classroom. This class includes 15 boys and 12 girls. Eight are white, 16 black, and three Asian. The class had just finished reading a chapter in the textbook. The teacher had prepared a transparency reviewing some of the points made in the chapter; he projected this on the front wall and told the students to copy it. Later in the period he replaced it with a second transparency. Another part of the day's assigned work was to answer the first half of the end-chapter review questions in the textbook. ("What is the key to successful computer usage?") The second half of the questions would be the night's homework.

While the students were carrying out these assignments, the teacher sat with me at the side of the room. Our conversation was interrupted by questions from students, by the teacher telling the class to be less noisy, and by administrative problems. ("You're late; where's your pass?") As in the lab, talking among the students is tolerated as long as it is not too loud.

Because of a lack of funds, there are not enough copies of the textbook to give one

Figure 7. Programming problem.

to every student enrolled in a computer course. Instead, the sections that are using the classroom in a given week get the textbooks, and when the groups switch rooms they hand over the textbooks to the other group. This arrangement makes it hard to keep track of which student is responsible for which copy of the book. Partly for that reason, the issue of bringing the book to class is given a good deal of emphasis; the rule is that a student who does not bring the book to class gets an F as his grade for the day's work. This happened to two students during the class I observed.

Interviews

For both teachers, the central point to understand about the goals of the computer facility is the tension between the official goals from downtown and their own private goals. They talk about the constraints imposed by the vocational education funding. For example, I asked the lab teacher about his own goals. (In this report I shall use "the lab teacher" to refer to the teacher I observed in the lab, and "the classroom teacher" to refer to the one I observed in the lecture classroom. But these are labels of convenience; both teachers actually teach in both settings at different times.)

Basically, I got involved with programming by doing it. And I think that the logic and the kind of thinking it requires from students is a good thing for them to get their heads into, since they have a real hard time dealing with things they can't finesse. You can't finesse a computer. I enjoy that. That doesn't fit in entirely with what [the school district] expects because of the fact that these are Voc Ed computers. We really can't legally teach things that are not related to entry skills and jobs. But for me, I like what you saw this morning, seeing them really puzzle with that number eight [problem], I think that's a really exciting thing.

In addition to disagreements with the district policy about goals, this teacher has also experienced the district policies as a source of pragmatic difficulties. I asked what fraction of the student body takes a computer course.

10% a semester, then a lot of the kids took it last year and then they prepared the Standards of Achievement and so last year didn't count or something so they're taking it again, so there's a certain redundancy.

The choice of textbook was also made by the district administration.

The teachers see an extreme difference (as did I) between the two halves of the course, the lab part and the classroom part. The lab teacher's goal of teaching logical thinking can be pursued in the lab, but not in the classroom:

Well, these courses actually have two sides to them. This side, which is hands-on, and that side, which is textbook stuff. The textbook stuff doesn't reflect that very much. It's really more of a vocabulary development course than anything else, because we don't have contact with a lot of the stuff that we study in the textbook, the more sophisticated data processing hardware concept. In here, in terms of the interest—Part of what motivates me is I like puzzles, and programming problems; you can really dignify it by talking about logic, but you're puzzling things out. And at this point—It's really hard for me to answer those questions; those answers aren't worked out. I don't think they're worked out anywhere. And they're not worked out for me. I got into this because I like to play with the toys. And so I drifted in this direction, and once you start playing with these toys, a lot of opportunities to continue to play with them open up because there are so few people who have any skill at all. Right now, I'm enjoying watching kids puzzle in a very rigid structure that the computer enforces upon them. And they have to learn the structure; they have to understand it enough to function in it. And sometimes the logic that you can see expressed in a flowchart that's pretty, and there's a certain aesthetic pleasure that carries over to seeing them start to get glimmers of how logic can be pretty. And that's exciting too.

I asked several students what they thought the teachers' goals were in the course. Most of them replied, as at the other schools, in terms of preparation for jobs. But two students said that the goal was to teach good program structure. "Right now they're trying to teach us how to program computers without using GOTOs. They're trying to show us the best way to program." "This year it's programming and some other software, but structured programming. We're doing it structured and not just going through it and making it look like a mess." I asked the lab teacher to comment on these responses.

We end up teaching BASIC only because microcomputer manufacturers make that available. And BASIC, as you know, is not a structured language. So when I first started teaching it, I wasn't too persnickity about this whole thing. And people had GOTOs out of their ears and they solved every problem by GOTOing someplace else. And

so now we're trying, as my experience has developed, I'm trying to make them relate to thinking ahead of time about what it's going to look like in order to do what they want it to do. And using escape hatches like that less often.

He added that he would like to teach an Advanced Placement Pascal course, but that the current equipment won't support Pascal and there is no budget for more sophisticated computers.

I asked him for his impression of the goals students bring into the courses. I was particularly interested to know whether the officially vocational nature of the program was reflected in the students' interests.

That runs the gamut. I told you, there are those that get here because the counselors can't think of anything else to do with them. There are those who get here because they think we're going to play games all day. And there are those who get here because they have fiddled around with programming and want to learn more. A lot of the kids in the computer programming courses, the second courses, are here because of that reason. When you see fourth period [an Introduction section] today, which has a lot of sophomores and a lot of people who just happened to be there at the wrong time or something, you'll see it's a different—Third period, first and third [Programming sections] are really nice classes. And you can see the interest and the intensity that they bring to them. Fourth period's a lot more flaky, because they don't really have any goals; they're just here.

I don't think there are a lot of them that are seeing a very clear, direct connection between what they're doing here and a job. I think they see a fuzzy connection, because everybody tells them that no matter what you do in 10 years you're all going to be doing it on computers. So there's that. In the other room we talk about some of the occupations, systems analyst and stuff like that. There are those that have those ideas in their heads. But I don't think people really see this as job entry. I think they may relate to the word processing part of it a little bit more like that. But still, it's really ambiguous. I don't think there are any direct connections.

The classroom teacher has a somewhat different perspective on the computer facility because he taught programming at this school, in the Mathematics department, before the current vocational program was set up. The early programming courses were taught using only the three timesharing terminals that are still in the classroom, but rarely used now. At first, very few students enrolled in programming classes. With 15 students each term,

for example, it was possible to handle the load by sharing the use of the three timesharing terminals. But as the public became more aware of computers, and the urge for “computer literacy” grew, the number of students enrolled in programming courses grew to over 100. That was when the Business department took over.

The funding came from the Business department, and they purchased all of the computers... And a lot of strings attached to the money. Number one, which is why you see the data processing book, we could no longer teach programming. In fact we had a running battle with the head of the Business department because once we started to teach programming... When we took the money we had to let anyone into the class. There were no prerequisites. Whereas before I had at least an algebra prerequisite. It was easier on me because the kids for the most part understood the concept of a variable. Whereas now I have a lot of students that don’t have the slightest idea. They’ll mainstream kids into this class that—For instance, this group is probably my most diverse. This group has kids that are GATE [Gifted and Talented Education] program all the way down to kids that just barely know how to read. It’s been interesting.

For instance, the programs that come with this textbook aren’t bad. It seems to be fairly well organized. The practical aspects, working with the computer, start out with just input and output kinds of things and even though you’re teaching them how to use a READ, DATA statement indirectly, they’re just putting things in and printing them, that’s it. And so probably the only thing I don’t like about the book is that it doesn’t let you build on the statements; it just sort of throws a bunch of statements at the kids. Whereas when I was doing it in the Math department I would build. For instance, I’d show them logically why in a counting loop, you can use a counting loop that transcends into a FOR–NEXT loop. Those kinds of things. I felt more successful, personally... And since we have to satisfy the Business department, I sort of backed off. I try to make programming for them as easy as possible.

This should be a semester course. It’s a year course. This district has mandated year courses. And I think it’s a crime. I think that for the number of students that are not interested by the end of the first quarter, let alone the end of the semester, and are forced to continue, it sort of—it makes a problem of discipline; it’s just unruly sometimes. I have two classes, my next class and my last class of the day, that are just totally unruly. It’s just a mess.

The only thing that’s really fit for everyone, I think—Word processing has been handy for everyone. And we keep the computer room open in the morning; we open at 7:30. We’re going to try to keep it open

during lunch. When we had just a programming class, and many assignments to do, the place was always full of kids. But now that we sort of dropped some of the intensity of the programming because there's so many kids, we don't get a full house over there like we used to.

I asked the classroom teacher to describe the goals of the computer courses as he sees them, and his own goals for the courses.

The basic goals of this class are really just literacy... The goals are to just expose the kids to computer operation, some software operation, in particular word processing. And data processing in general. This textbook is much more complex than what we need. We could do this all in a semester. And we would probably be able to expose more of the student body. We're doing a little more than 10% right now, and that's horrible for this day and age. Especially, like I said earlier, with the kids who aren't interested, get 'em out, crank 'em out in a semester.

I'm not a game type person, and that reflects in the way I handle the classroom. I don't like to do all these cutesy things with computers just to keep the children interested... I'm an applications person; I like to give them problems and have them solve them. I just don't like to play games. Those are basically the goals in this class. And for those kids who are unable to meet them, for instance I'll give them a program for them to do, and let them struggle with it; maybe two, three, four kids in the class will get it, be on top of it, and then what I'll have them do is help others work it out. Sometimes the help is just accomplished by copying. Or I'll go over the program in detail on the board, have them copy it down, and then their basic task is to just type it into the computer, run it, print it out.

I asked students in the Programming class their reasons for taking the course. One said "I want to be a computer programmer." Two others said "because I like computers" and "for fun"; these two are also interested in computer-related jobs. Two others had specific careers in mind that are not primarily computer related, but feel that computer experience will be helpful in those fields: "Originally I planned to go into the computer field, but that was two years ago... Even what I plan to do—I want to be a dentist—even then you have many records, so you need computer skills." "When I get out I'm going into the armed service, and what I'm going into is water purification, and it has a lot to do with computers." None of the students mentioned entry-level jobs in the computer field as an objective.

Summary and Conclusions

Of all the schools I visited, this one shows most strongly a conflict of goals between the teachers and the administration. At school C there is some conflict within the teaching staff, but there is no imposition of goals from above. At the other schools, the teachers *are* the administration: the district goals for educational computing are set by committees of teachers or by a teacher who is also the district curriculum coordinator for computers.

This school offers two year-long courses, both with explicit, official vocational goals. The ultimate source of these goals is the state government, through its guidelines for the use of vocational education funds. The details were set by school district administrators outside this particular school. These details include the year-long format and the specific textbook used in the courses. In the interviews, both teachers indicated that they do attempt to carry out the district's goals, but also that within the limits of those goals they are most eager to pursue their own, quite different goals. Both are interested in problem solving, and for that reason they emphasize actual computer programming as much as they are allowed to.

The teachers' sense of constraint was apparent throughout my visit. They have done, under these circumstances, an impressive job of pursuing their educational goals. For example, it was the students at this school who most clearly understood structured programming and problem-solving to be a goal for the faculty, although that goal was expressed to me by teachers at every school.

Does the district's vocational training goal reflect the actual needs of the students? None of the ones I interviewed suggested that they intended to pursue an entry-level job in data entry or word processing, for example. The ones who mentioned jobs either want to be computer programmers or have some other career in mind to which computer skills are incidental.

The laboratory work in both courses excites the enthusiasm of the teachers and of at least some of the students. The textbook material does not. I think the computing faculty here is to be commended for emphasizing the hands-on part of the course as much as they do.

School E

Description of the Facility

This is a school of about 2800 students in a middle class, multiracial town. The Mathematics department offers courses in computer science, and the Business department offers vocational courses in word processing and computer typesetting.

There are more than 50 microcomputers in the school. Most of them are located in a cluster of three adjacent rooms, although some are in other parts of the school for special purposes. The Mathematics department operates two computer classrooms with a connecting door between them. One, the “literacy” room, contains 15 Commodore 2001 systems using shared floppy disk drives (in two clusters). This room is used for the course called Personal Computing and You, the lowest-level computer course offered in the department. The other Mathematics computer center is the “computer science” room, used for the three higher-level courses. This room contains 12 IBM PC systems. Both rooms have the computers at one end and regular classroom seating at the other. On the walls are demonstration pages from various brands of computer printers, advertising posters, and posters for the school’s Outing Club, also headquartered in the computer science room. Outside the door of the computer science room is a hand-drawn sign proclaiming it to be the Neener Room; a “neener” is the local name for someone obsessed with computers.

Next door, but not connected to the Mathematics rooms, is the Business department’s computer facility. Its door is part of a cheerful mural full of animals and vegetation. In the middle of the colorful door is this sign:

STUDENTS: You may not come into room 21 for any reason whatsoever unless you have a visible pass from a teacher of WORD PROCESSING, TYPESETTING, COMPUTER PROGRAMMING, and or a teacher you are T.A.ing for.

A second door connects this room to the Teacher Curriculum Center, off limits to students, where the teachers’ mailboxes and other facilities are. (The Mathematics office, for example, is a small room inside the Curriculum Center.) When Word Processing and Typesetting

classes are not in session, teachers use the computer facility for their own word processing needs. The facility is also used by the yearbook and the school newspaper. This room contains 16 Commodore 8032 systems sharing disks and printers in three clusters. One of the three printers is a letter-quality impact printer (a Spinwriter); the others are dot matrix. The room also contains four IBM MT/ST (Magnetic Tape Selectric Typewriter) typesetting machines and an Addressograph/Multigraph Comp/Set 3510 phototypesetter. The latter uses 8-inch floppy disks; one of the Commodore machines is connected through a serial port to a Sheffstall 8-inch floppy disk system to allow file transfer between the Commodores and the Comp/Set.

Another cluster of six Commodores is in a room attached to the library and media center. This room is available for informal student use all day. A sign on the wall forbids “games requiring physical agility”; that is, adventure games are allowed but arcade-style games are not. Students can use this room for word processing. Students in the Personal Computing course also use it if they want to spend extra time on a programming assignment.

The Business department has two teachers who teach computer-related courses as part of their assignment: one for Word Processing and the other for Typesetting. The Mathematics department has five teachers for computer courses: three who teach the Personal Computing course and two who teach the more advanced courses. The department teaches a total of eleven sections of computer courses each semester.

Courses Offered

Introduction to Word Processing is a semester course in the Business department. It has Typing as a prerequisite. Two sections are offered each semester, with about 30 students in each section. Appendix C.8 shows the official goals for the course. (The school publishes a Course of Study book for each department. For each course, the book includes a summary of the goals and a unit-by-unit curriculum guide. These guides are too long for me to include here, but Figure 8 is a sample, the first of four pages for the Word Processing course.)

Typesetting is a Regional Occupation Program course; that is, it serves students from

Figure 8. Sample curriculum guide.

surrounding districts as well as local residents. It is a semester course, meeting two consecutive periods (two hours) each day. Its goals are shown in Appendix C.9. A second semester course, *Typesetting Lab*, provides for additional experience in this area. (The Business department also offers ROP courses in offset printing, so the material typeset in this computer facility can be printed within the school.) There are six Typesetting students this semester.

Personal Computing and You is a semester course offered by the Mathematics department. It is what the department considers its “computer literacy” course offering; it is open to any student with at least eighth grade skills in reading and mathematics. The course teaches BASIC programming through self-paced units. Its goals are shown in Appendix C.10. Six sections are offered this semester, each with 28 students.

Computers in Mathematics is a continuation of Personal Computing, teaching more advanced BASIC features. It is a semester course. It also requires Algebra as a prerequisite. The emphasis in the course is on using the computer as a tool to present mathematical ideas. The course goals are in Appendix C.11. One section of 25 students is taught each semester.

Introduction to Computer Science is a full-year course teaching the Pascal programming language. Geometry is a prerequisite for the course, partly because it teaches logical thinking through formal proofs and partly because it is a course in which students learn to handle heavy homework loads. Before this year, the course was taught using BASIC. The official statement of course goals has not yet been changed to reflect the new Pascal-based curriculum, so it is not included here. Three sections of this course are taught each year, with 25 students in each section.

Advanced Computer Science is a full-year Pascal course that prepares the student for the Advanced Placement Examination in Computer Science. It requires Introduction to Computer Science as a prerequisite. (This year’s Advanced students are meeting Pascal for the first time. Next year, when the Advanced students will have a year of Pascal experience already, the department wants to reduce the four semesters to three.) The course goals are in Appendix C.12. One section is offered, with about 25 students each year.

Classroom Observations

The two rooms operated by the Mathematics department are open for student use at 7:30 each morning. (Classes start at 7:51.) I saw a few students come and go in the computer science room, working on programming assignments or word processing. The Business computer classroom is not open for student use before school, but students can do word processing in the library.

Word Processing

I observed a Word Processing class with five boys and 17 girls. This is a vocational class, and most students who enroll are interested in developing a job skill. The Word Pro program is used on the Commodore computers. The teacher is an experienced member of the Business faculty, who also teaches typing classes, but she is teaching word processing this year for the first time.

The class began with the students in chairs facing the blackboard at the front of the room. The topic for the day was the global search and replace command in Word Pro. The teacher introduced the topic by remarking that it is “unique to word processing,” that is, it is not something one can do with a conventional typewriter. She gave examples of using the command, and then invited the students to go to the computers and try it out. Since there are more students than computers, not all can type at once, so some of the students remained in the front section talking while awaiting their turn. One group of five computers was unusable because of a problem loading files from their shared floppy disk drive. The teacher sent some of the students who were trying to use these computers to the library. She commented to me that hardware problems are very common, although not usually with an entire cluster at once.

The students work on typing exercises from a textbook like those used for typing courses, but tailored to word processing. The exercises are examples of various kinds of business letters. Students share textbooks, because the department does not have enough copies for an entire class. (The book is out of print, and the department has been unable to find another book using this particular word processing program.)

Typesetting

I observed only a small part of a Typesetting class, because of a scheduling conflict with a Computer Science class. The course begins with the use of the IBM MT/ST equipment. IBM discontinued this equipment line many years ago and has recently announced that service contracts will no longer be available for it. The school has therefore decided to keep the machines until they break and then abandon them. The second part of the course uses the newer and more sophisticated Comp/Set, a phototypesetter with a range of different type styles and sizes. (According to the typesetting teacher, even this machine is obsolete, but it can still be serviced.) The class goal for this session was to enter files using the Commodore computers and then transfer the data to the Comp/Set. I watched the teacher with one student at the Comp/Set keyboard, dealing with lines in the data file that were too long for the typesetter to handle. It was frustrating for them because they ended up deleting characters and manually re-entering them on the next line.

Personal Computing and You

I attended one session of this introductory BASIC class. There were 13 boys and 10 girls present. The class started with the students seated in front while the teacher gave a brief explanation of the day's work. The course is divided into units, and the day I attended was a transition day between units. The teacher explained that each student could work on the programming assignment for unit three or the enrichment activities (to be explained shortly) for either unit two or unit three. He concluded by telling the students that they were to log off the computers by five minutes before the end of the period.

This computer center uses the same Commodore shared-disk hardware I had seen at school D and in the word processing room at this school. However, a software package called Regent is used in this center to coordinate the use of the disks by the clusters of students. Instead of calling out "Don't use the disk" before inserting one's own, personal floppy disk in a drive, the students in a cluster share a single set of permanently mounted disks. Each student logs in with an identifying number and a password, as in a timesharing system, to gain access to his or her own files. This arrangement avoids some of the confusion

of other Commodore installations. The cost of its convenience is that when a computer malfunctions, the students who are scheduled to use it cannot be switched to a machine in another cluster, because their files are on the communal disks of their own cluster.

Students work on programming assignments in groups. The groups are set up at the beginning of the semester and may contain between two and four students. (By mid-semester, some groups have become groups of one because of course drops or other rearrangements.) Students are allowed to choose their own groups, although the teacher I observed recommends to his students that they not choose their best friends as partners. Each group works on a program together, and receives a collective grade for it. Other components of the course (tests and enrichments) are graded individually.

On one wall of the room is a large array of mailbox-like cubbyholes containing course handouts. Students take the materials they need, as they need them. The first handout is an introduction to the course (Figure 9). This introduction has a slightly different statement of goals from the one in the official curriculum guide (Appendix C.10). It also shows the division of the course into units and explains the various types of assigned work. Another introductory handout shows the behavior rules for the course (Appendix C.13). This sheet combines academic concerns (“DO come after school for help before you fall too far behind”) and computer-specific concerns (“DON’T bend, fold, or mark on floppy disks”) with more general disciplinary rules (“Students will keep hands, feet, and objects to themselves and quiet”). The rules, although written specifically for this course, do not completely reflect the actual student behavior encouraged by teachers; in particular, the rules prohibit talking without permission, but in practice there is constant talking within and between groups.

Each unit begins with a worksheet. (Despite the name, it can be several pages long.) A worksheet begins by explaining its objectives (“you will be able to...”) and then leads the student through an interaction with the computer. (See Appendix C.14.) A typical worksheet, like the one shown here, starts with an already-written program and asks the students to predict its results. Then it asks the student to modify the program to achieve a stated effect. Each worksheet ends with a review section. There are unit tests that ask questions similar to the review questions in the units.

Figure 9. Personal Computing Course Introduction.

Figure 9. Personal Computing Course Introduction (continued).

Each unit also includes a programming assignment. The assignment may contain several programs. For example, the one shown in Appendix C.15 contains two programs required for every group plus a choice of three programs, one of which must be done by the group. An important part of these programming assignments is the “modification”: after a group completes a program, the teacher tells them to modify it in a particular way. He chooses the assigned modification to match the ability of the group members; it can be as easy as adding comments identifying parts of the program or it can be a major change to the program structure.

The most unusual part of the course is its use of enrichment projects. These include both non-programming (Appendix C.16) and programming (Appendix C.17) suggestions. Students may also propose enrichment projects on their own. The enrichments are an individual activity, not part of the group work. What makes the enrichment work unusual is the role it plays in grading the course. In order to get a C for the semester, a student must average 75% on all assignments and tests. Beyond that point, though, the numerically graded work does not contribute to the overall grade. Instead, a student must complete three enrichment projects for a B, and six for an A. (Three of the six must be programming projects.) Enrichment projects must be negotiated with the teacher in advance. The result of this policy is that even the least able students can earn a B, but the teacher can steer the better students into more challenging projects.

During my observations, the students worked on their group projects without direct supervision by the teacher. His role was to answer questions, both technical (why doesn’t this work) and administrative (did we get credit for this assignment). A teaching assistant, a more advanced programming student, was also available to help the groups with their projects. The groups worked with a good deal of side conversation, but also with steady effort and with very little confusion about what was expected of them.

Computers in Mathematics

Computers in Mathematics is a second semester of BASIC programming for students who have taken Personal Computing and want to continue. It is taught in the computer science room, using the IBM equipment, but it is not considered part of the Computer Science (Pascal) sequence. Only one section of this course is offered each semester. The class I observed contained 11 boys and 10 girls. Algebra is a corequisite (listed as a prerequisite in the course of study, but not enforced as such). The algebra requirement is one factor that limits the number of students who continue after taking Personal Computing, since many of those students are not in the college-bound math sequence.

On the day of my visit, this class was taking a test (Appendix C.18). I therefore did not see students actually using the computers, and I was limited in my ability to interview students during the class. The unit that had just been completed was about the application of computers to elementary number theory (primes and factors). Later topics include approximation and transcendental functions. Programming topics introduced in this course include arrays and random number generation.

Introduction to Computer Science

Three sections of the introductory Computer Science course are offered each year. (It is a year-long course.) The course is new this year in its present form; in the past, a course with the same name was taught using BASIC, but the new course is based on Pascal. The section I observed has 12 boys and six girls. Geometry is a prerequisite for this course, so the students are mostly juniors and seniors.

The period began with a quiz about binary numbers. The quiz required students to convert between decimal and binary, to perform subtraction problems in binary, and to calculate the number of different characters that can be represented in an eight-bit byte. Binary numbers are part of the curriculum not because they are needed to understand computer programming, but because they are used as an example in this unit; the programming assignment that students were about to begin is to convert between decimal and binary automatically (Appendix C.19). Decimal numbers are represented in this assignment by

ordinary Pascal integer variables; binary numbers are represented as a sequence of characters, each of which is a zero or a one. The purpose of the assignment is to motivate the introduction of arrays, which the class had not yet learned. A binary number would best be represented as an array of characters. Instead, the programs must be written using a single character variable and changing its value in a time sequence, generating or accepting one digit at a time.

This course uses both textbooks and handouts. An introductory handout (Figure 10) explains the principles and structure of the course. The course is somewhat similar to Personal Computing in that students program in groups, and that group projects, homework, and tests all contribute to the grading process. Appendix C.20 is an example of a programming project assignment, and Appendix C.21 is a sample homework assignment. One difference between this course and Personal Computing is that here the grade is based primarily on tests; only 20% of the final grade is based on programming projects or other in-class work.

During the discussion I observed about the decimal-binary conversion assignment, the teacher tried to elicit suggestions from the class about the algorithms to be used. For the most part, the students were not very responsive. Most seemed unwilling to expose their thoughts to the group if they were unsure of the solution. They did participate in class discussion at moments when they were on more solid ground.

Advanced Computer Science

Only one section of the Advanced course is taught. The course is directed toward preparing students for the Advanced Placement examination in Computer Science; this goal determines much of the curriculum. The course style and structure are much like those of the introductory Computer Science course. (Appendix C.22 is the introductory handout; Appendix C.23 shows the second semester curriculum schedule.) It is a year-long course, although that may change in the future. This year's students are learning Pascal for the first time, but next year's Advanced students will already have had a year of Pascal. The class includes 12 boys and two girls.

Figure 10. Computer Science course introduction.

Figure 10. Computer Science course introduction (continued).

Most of the class session I attended was devoted to review for the A.P. exam, one week away. The teacher went through an outline of the topics included in the examination, filling gaps in what had been taught earlier in the year. For the most part, the gaps were in technical vocabulary. For example, the students were unfamiliar with the terms “formal parameter” and “actual parameter,” although they had no trouble understanding the definitions. There was also some discussion about the mechanics of signing up for the exam, and how to decide whether or not to take it. (Most, but not all, of the students in the course should take the test, according to what the teacher told them.)

The teacher also assigned a programming project: simulate rolling two dice 10,000 times, and record the results by drawing a histogram with the axes properly scaled and labelled. This project was to be done as homework, on paper.

Interviews

I talked with several teachers and students informally during idle moments in the classes I observed. I also formally interviewed one teacher, the one who teaches the Advanced course and two of the introductory Computer Science sections, and is also K-12 computer curriculum coordinator for the school district.

The Word Processing teacher and the Typesetting teacher are both experienced teachers in the Business department, but relatively new to computers. (This is the first year for the Word Processing teacher.) They both, therefore, inherited a course structure set up by someone else, so they don’t always know why a decision was made one way and not another. But they do not merely follow tradition unthinkingly. For example, the Word Processing teacher has replaced the first unit of last year’s course, about the history of word processing, with a unit on proofreading using another textbook. She considers this more relevant, and also helps restrict the course to serious students by starting off with something substantive. Both teachers have straightforward vocational goals for their courses. The students’ goals are also essentially vocational, although not necessarily with a particular job in mind.

The Personal Computing teacher I observed is one of three who teach sections of this course. He describes the course goals in terms of computer literacy and also in terms of

providing an appropriate experience to each student in a very heterogeneous group. He showed a strong awareness of the individual strengths and weaknesses of each student. In some ways the course structure is like that of the Open Classroom concept, and to make such a structure work demands much more preparation by the teacher than more traditional teaching styles. I asked about the group work. He said that he recommends to students that they form groups of three, choosing colleagues who are not close friends, but that students rarely take this advice. Instead, they work with their friends, and then discover that it takes a special effort to discipline themselves to get the work done instead of being sidetracked. Students must also learn to work independently, he said. At the beginning of the semester he puts much of his effort into teaching how to find handouts, how to know what to do next, and similar planning skills. But the students do learn these skills by mid-semester.

I asked whether he considered things like self-discipline and independent work skills to be primary goals of the course, or whether they are simply things that students must master as part of the process of learning the *real*, computer-related goals. The answer was that these skills are valuable in themselves, but that the course was not set up with such learning as an explicit goal. Instead, the autonomous style of work in the course developed out of the need to accommodate students who work at different rates and with different degrees of programming ability.

The reasons given by students for taking the Personal Computing course include both fun and learning job-related skills, as well as “the counselor put me in this.” Some students speak of themselves as having considered a computer-related career, but having learned from the course that “I’m not smart enough.” (I heard no bitterness in such comments; even these students approached the course with energy and good cheer.) Several students referred to the teacher of this section as the best teacher in the school.

The teacher I observed in the Computers in Mathematics class also teaches one section of Computer Science. (This year he is, therefore, the only teacher with classes in both BASIC and Pascal.) I asked him about the goals of Computers in Mathematics. He said first that it exists to fill a gap between Personal Computing, with no prerequisites, and Computer Science, with a Geometry prerequisite. Students who have taken and enjoyed

Personal Computing, but who are not eligible for Computer Science, need a computer course they can take. The particular design of the course, featuring mathematical problems, was chosen to motivate students to continue in mainstream mathematics courses, and also to motivate the computer programming by showing a real use of computers. He finds it a pleasant course to teach because students take it mostly for fun; the ones who want to be professional computer programmers take Computer Science, while the ones who are looking for lower-level job skills stop with Personal Computing or take courses in the Business department. I was only able to speak with two students, who finished the test early, but they agreed that enjoyment was their main reason for selecting the course.

In a formal interview, I asked the Computer Science (introductory and advanced) teacher about his goals. He distinguished his goals as a teacher of computer science, his goals as K-12 curriculum coordinator, and his own personal goals.

As a computer science teacher my goals are to teach problem solving utilizing the computer. And so it's to give the students a variety of problem-solving skills that they can utilize. Of course part of that is learning the language, the syntax and all that. But the primary goal is to teach the problem-solving skills. That's the overall goal.

Curriculum coordinator. There's a number of different objectives that I have. The primary objective is to utilize the computer for fulfilling objectives that you couldn't fulfill without the computer, so as not to use it as a worksheet, and so to use it for things like simulations, for motivational type work—I often feel a lot of word processing is more motivation, in terms of using that to teach English, that type of thing. So to infuse the curriculum areas with computers, not just to give them computers, but to help them utilize the computer as a teaching tool, to do things they couldn't otherwise do... We're currently investigating how to utilize the computer for computer-managed instruction, for an objective-based education. So that's a goal.

There's a goal to teach—well, one, I love the theory of it. And it really makes me happy when I get a student that I've had for two years that comes out of the Advanced Placement class, and in a sense it makes me happy if they're bored with programming. They're no longer that excited to get on to the computer just to see the flashing lights and all that. They are more into receiving a problem, and enjoying the problem for the sake of solving it. And once they've solved it then the programming part of it is really minor. And so I enjoy that excitement over the theoretical.

I asked if he sees a connection between the computer science goals and the general curriculum goals.

Oh, I see, how do I see those two as together? I see it as another subject area, and I see it as what I feel is the most interdisciplinary subject there is. I feel that in that class, well as you saw today, teaching logic, writing, and the most exciting thing to me is to be in the Math department and to be teaching writing as much as I am. And to be teaching problem solving that's applicable to social science, to English, to whatever, in terms of breaking down a problem and that type of thing. So I see it as a subject that is both separate, in terms of computer science being a different science from physics, something essentially in the Science department, and I also see it as a logical exercise that is beneficial in all subject areas.

I asked how the goal of problem solving is reflected in the structure of the Computer Science classes.

Each unit ends with a project... It always works out this way, that for two, three days the students are kind of taking the problem and rolling it back and forth in their minds, and it's often a difficult thing for the teacher that wants to move on to the next topic, and says, "Why don't you get it, already?" But that process is the process of, "I have no idea how to solve this," bringing that down to actually solving the problem is an important skill... So the projects are really putting those skills into practice, into use. Also the way that the grading system works, that tests are, depending upon the class, either 60 or 70 percent of the grade, indicates to me that the project is also not the end goal. The end goal is to learn the material, to learn the skills, and to prove it on the test. And it used to be that projects were 70%, and I realized that all that I was measuring at that point was how much the kids worked at the computer, how much they were willing to document, how pretty they could make it look, and not whether they were learning the material. And the other thing is that projects are self-motivating; I could make them 1% and they would do them.

I asked him to explain further why he considers emphasizing tests in grading a better way to promote problem solving.

I think it's a lot better, because I can say to a student that wants to learn arrays on their own before, like some of the students, in that Pico-Fermi project [a number guessing game program] arrays is really the way to do it, but we hadn't learned them yet. Students that have read ahead, if I made the projects count so much that they were

fearful of not finishing the project and then getting a bad grade, that would stop their learning. So it allows them to take a step beyond, and take a chance, which is a good thing to be able to do in computer science. Also it takes a lot of pressure off of me. A disk goes bad, this happens, that happens, a student is absent, I can say, well, you didn't finish it, we'll grade it on that, and I don't have to worry a lot about it. Another thing about it is, grading projects is extremely time-consuming, especially if you've got the pressure that it counts a large part of the grade. I learn a tremendous amount from grading projects. I mean, I grade 20 projects, and every one is different, and I feel like I can solve this problem forwards and backwards. But it doesn't do any good for my students when I do that. And so by making the test count more I can give them the different methods.

Don't you think, I asked, that emphasizing the tests gives students the idea that what counts is right answers?

No, because the tests are—There's always essay questions, there's always a program, that have several different answers. And so, like this girl coming up to me at the end of class today: "Do we have to memorize the ASCII chart?" And I point out to my students when I want them to memorize something, like the syntax of a function and where the colon goes and all that. But they always want to memorize things, like on the sorting test, "Oh, I stayed up all last night memorizing this," and they memorize it line for line, and I say "I don't want you to do that." And so I try to teach the theory.

I asked to what extent he thinks his students are aware of his problem-solving goals.

That's a good question. I do bring it up. It would be an interesting thing to poll whether that is getting across... I give my students a lot of homework that they don't try on the computer. And whenever I tell that to computer science teachers a lot of them say "That's strange," but when you talk to university people they say "That makes sense." And I try to reinforce at that time that it's not necessarily trying it on the computer that's the point. I think in my Advanced Placement class nearly all of them know that. I would say in this class [Computer Science] fewer of them know it because when you're starting off teaching Pascal there's so much syntax and there's so much that's just basic stuff that I really do want them to know, and so I think that the further we go, the more they'll recognize these goals.

I asked about the relationship between the Computer Science (Pascal) courses and the literacy (BASIC) courses. Are they considered part of a single sequence?

No. There's basically two tracks, is the way it works out. There's Personal Computing, which is for anybody with eighth grade skills in math, reading, and then Computers in Math which is a second semester if they want to go on and study the applications of BASIC to algebra or pre-algebra topics. The Personal Computing class essentially gets up through FOR-NEXT loops. And it's really covering things on a very, very kind of basic level. And Computers in Math is in many ways much the same type of thing. They learn more about BASIC but really it's the application that we're speaking of there. And neither of those is prerequisite to Computer Science. So there's first year Computer Science and second year Computer Science.

I would say that [of] people that are in Computer Science, 30% of them got into it through taking a literacy course and saying "it's kind of fun, I'd like to take the other one." Since it is in the Math department, and all of us came from a math background, we are kind of slanted towards trying to interest them in math, and so there is a geometry prerequisite to Computer Science. Not just for the reasons of mathematics but also for the logical thinking skills, and that they've made it through a course where they have homework every day.

I then asked about the process by which curriculum decisions are made. How was it determined that there would be literacy classes in BASIC and Computer Science classes in Pascal?

It's been a continual process. We have never taught one year the same as the next year in terms of courses. We started off—well, how are the decisions made. The decisions have been made by basically about three of us that are in the Computer Science department, you know, seeing the trends and seeing what we have available equipment-wise and software-wise and all that. There's certainly a lot to be done, especially in the area of bringing it out of the Math department. But just to give you a feel for how it all progressed, five years ago we started the Computer Literacy course and Computer Science in BASIC. Then we taught that for—actually, we did have something the same, we taught that for two or three years. But we were continually developing materials, and that's what that book [the handouts for Personal Computing are to be published as a textbook] came out of, all that time where there was no textbook for somebody with eighth grade skills in BASIC programming. And so then one of my colleagues developed the Computers in Math class, and then I developed the Advanced Placement class in Pascal. And then this year we changed the BASIC Computer Science class into a Pascal Computer Science class. So then next year's Advanced Placement class is going to change because they'll be taking off from the middle of

Pascal... So that's all been left to the teachers. It has not come from the principal or the district, or anybody else.

I see a change coming in the Computer Science sequence. Just because we can certainly condense it to three semesters instead of four, because of the changes in doing it in Pascal all the way through. And so we'll probably create an Assembly Language, or an Advanced Topics in Programming type of class. We also, by the way, have an independent study computer science class, which we've got three students in now that have taken the AP test last year. They all learned dBase [a data base management program]. And they did some dBase programming for the administration. They've basically been on assignment to write programs for teachers and administrators.

So I see that changing. I see our literacy course changing to encompass less programming and more word processing, applications, types of things. I see the Computers in Math thing as an extremely valid class, but I also see Computers in Art as an extremely valid class, and those I think would be the essence of it. And also Business classes that utilize computers.

I asked if reducing the programming emphasis in the Personal Computing course would lead to a more extreme feeling of separation between the two tracks.

Well, actually, there's been talk of making the literacy class into a real literacy class, and then having a course in BASIC programming, which would be available to the same students. There is a kind of a tracked feeling because there's different types of kids. And I think it's real valid to offer a class which moves along quickly; I don't think that we need to offer Pascal to people that oughtn't to be able to handle it, essentially.

I asked his opinion about the goals his students bring into Computer Science courses.

The CS courses I would say over 50% are going to be either Computer Science or Computer Engineering majors, or at least that's their goal. (I would say 100% are going, or 95% are going, on to college.) I would say another 25% are going to be math, science, engineering related, and the rest of them are an interest, or a feel like I should, but primarily they're there because they want to be there. The literacy course, there's certainly an amount of interest, curiosity, there's also five math credits for a class which I feel is very easy, and I don't know how—still students get Ds and Fs, I don't know how they do it, they really must work at it. But we lack—I don't know if you know about the new high school requirements for math, but they're going up to three years, the requirement, and for the student that is not going into Algebra, there's not a whole lot to take. And so there's that part of it too.

I asked about students using the computers on their own, outside of formal courses.

It's interesting if you come in the middle of a project, because you'll find the room is a zoo. A lot of slowdown in AP because we are in the review phase instead of in a programming type of phase. Normally throughout the whole year there's always some kind of project that's going to be due soon, so they're continually working, but right now they're just studying. And those kids have a tendency to hang out around there.

I encourage the Advanced Placement students to come in during fifth period, when the room's available. I also recognize that most of these kids have tremendous loads and so I really say, you know, if you need to, go ahead and do it. And we keep the room open after school essentially every day. And I'm there before school every day, although there's not that much time once everything gets going. And computers are available in the library. In the literacy course, that room is also open every day, staffed by one of the three teachers that teach that course. And so students are encouraged and generally the times that they come are when a project is due and they need more time.

I pointed out that he was talking about computer use outside of class hours, but still course-related. What about students who aren't in a computer course at all during some semester, but come to use the equipment anyway?

We have a few of those. Those are "neeners." We have a few of those, not as many as we used to. Partially I think it's because it's become more of the establishment. It's become more of the established course work and, like, the student that used to come by and say "I really want to learn blahbity-blah," and we didn't teach it, and I'd say, well, here's a book, come by after school and we'll go through it. And partially it's—there hasn't been an extracurricular computer club, which would certainly do that type of thing, and that's just because nobody has had time to run it. And also those things take the form of extra credit. A student wants to learn more, and so we do it as an extra-credit project, and it's usually more related to the course. I mean, I've got a few students that—there are a few students that come in just to do their own thing. There's a blond, the one that I asked what he was doing in class. He's in all the time. He's an exchange student from Sweden, actually, in there all the time. And the guy across from him was using some—I think he was in the Computers and Math class, I think he uses a word processor on the IBM. You'll definitely want to head over to the library to see other kids working on their own stick. I only see, really, the computer science students now. And the literacy students, well one thing that's

interesting in the way the literacy course is run is, you need 75% on tests and quizzes for a C; beyond that tests and quizzes don't count for anything. You need to do three projects for a B and six projects for an A. And the projects can either be the ones that we suggest or they can make up their own, and we try to encourage them to make up their own. So in that class, we try to get them to write their own thing.

I asked if social or ethical issues arise in the curriculum.

It's certainly an issue practically in terms of copying software and all that. And one of the things that I'm going to be working on in the next year is a master plan for the district, and so a lot of these things will come out. That's something that should really be addressed in the literacy program, and it's probably addressed for about 15 minutes, one day. It's something that is addressed in the computer classes primarily when a student wants to take home their disks, or something like that, where they want to copy their programs at the end, we say we can give you your programs, but we can't give you the Pascal system. But it is not covered that much. It's something that we need to work on.

Finally, I asked whether students use the computers to play games.

Yes they can. You know, that used to be so much more of an issue than it is now. I never have kids ask that any more. Probably because it's become so serious and so much—that's a part, and maybe because in the very beginning, when we first got the computers and games were the in thing, there were only certain times that you could use games. I mean, it really was a problem when we first started. And certainly when a project was due—The rule was that if people were waiting for a computer, you could be kicked off if you were playing a game. And so I have some students that play games during lunch, but not too many. It's certainly not advertised, is one reason, or let's say really condoned, and I don't talk much about games in class. Probably should a little bit more, just to get that interest.

Summary and Conclusions

This school has extensive facilities and course offerings in computer education. The Business department and the Mathematics department both operate computer laboratories and offer courses, from vocational training to advanced computer science. Unlike school A's single sequence, the courses here are organized into separate tracks. Three computer centers are kept busy with courses most of the day.

A district-wide policy statement on computer education is being planned, but for now the school's official statement of goals is in the individual course descriptions in Appendix C. These official goals are numerous and wide-ranging; for example, both "examine the applications of computers and their impact upon society" and "code fluently in a well-structured fashion using an accepted high-level language" are among the goals listed for the Advanced Computer Science course. The goals discussed by teachers in interviews are primarily job training in the Business courses, computer literacy in the BASIC sequence, and problem solving in the Pascal sequence.

A common remark near the end of presentations at computer education conferences is, "Thank goodness computers are a new thing and there's no standard curriculum. That's what allowed me the flexibility to set up this innovative program." Many computer-using teachers look forward pessimistically to a time when Computer Programming will be a standard course of study, akin to Trigonometry.

I have seen the future, and it works.

At this school, the computer curriculum is so well-established that it has even been divided into three distinct tracks: vocational (Business courses), literacy (BASIC courses), and computer science (Pascal courses). The curriculum for the literacy sequence, developed at the school, is to be published commercially. Courses at all levels run extremely smoothly, despite occasional hardware failures. Students in all three tracks seem cheerful, relaxed, and studious. "It's become more of the establishment."

I was particularly impressed with the Personal Computing course. In my own teaching, I have struggled with the problem of tailoring one introductory course to the needs of a strongly heterogeneous student group. The structure of this course seems to be a workable solution, not only allowing the needed flexibility but even performing the paradoxical feat of using grades as an instrument to promote student autonomy. The structure places severe demands on the teachers, and they seem to live up to those demands.

Two things bother me, though, about this school. The first is a pervasive sense of blandness and tight control. I started having that feeling when the first class began promptly at 7:51, not 7:50 or 8:00. Other administrative clues are official course descriptions with 74 specific objectives and written behavior policies for each class. (These rule

sheets are required by the principal of the school as part of a uniform approach to school discipline.) The advanced programming students can pass a college placement examination, but they are not passionate about computers. The school trains good academic soldiers.

I do not mean to suggest that there is an atmosphere of 1984-style regimentation in this school. I felt no fear or resentment among the students. Perhaps a more apt literary reference would be to the cheerful but passionless inhabitants of *Brave New World*. (I felt the force of this analogy quite strongly in the Personal Computing class while talking with a student who was neither embarrassed nor resentful when he described himself as not smart enough for Computer Science.)

The student newspaper, which I first read as an example of the typesetting facility at work, says that a survey of students indicated that they all cheat. So under the skin, they are real teenagers. Over the skin, though, one doesn't see much purple hair in these corridors. And in the computer lab, students come to program the computer when they have a project due in a course, and not otherwise. (I do not mean to suggest that every healthy teenager must present a rebellious image, nor that all must be passionate about computers. But in both cases I am surprised to find *no* such students in a large high school.)

My second reservation about this school's computer courses is the extreme eclecticism they show. The faculty does a great deal of effective planning to achieve the goals of each course, but there is no sense that some goals are better or more important than others. Parallel sequences in two different programming languages are an example. To leave no stone unturned, the school plans to institute an even less technical "literacy" course (although 80% of the school's students already take at least one computer-related course before they graduate), and to explore computer managed instruction in all curriculum areas. (Computer managed instruction, I think, is another sign of the administrative preoccupation with control.)

One problem with so many parallel goals is that the school risks a dilution of its efforts. At many schools, this problem manifests itself in the form of a shortage of computer equipment. It may be impossible to provide a literacy course to every student and still give more advanced students enough computer time to pursue complex programming projects.

Here, so far, there is enough equipment. (But that may be partly because advanced students are discouraged from programming with a computer. They are told to write programs on paper and turn them in without running them, and they are encouraged to be “bored with programming,” so that they do not demand computer time for extracurricular projects.) I am not sure that there is enough teacher enthusiasm and student enthusiasm, however, to support this curriculum plan.

A more important objection to eclecticism is that there are real conflicts in the messages taught to students. In the Computer Science sequence, students are given a handout that says, “It is *harder* to learn Pascal if you know BASIC than it would be if you did not know how to program at all.” This sentiment agrees with that of many introductory Computer Science instructors at universities. But the faculty does not heed its own warning in planning the more elementary programming courses. Similarly, in the Personal Computing course the grading policy emphasizes programming projects over tests in order to promote problem solving; in the Computer Science courses the grading policy emphasizes tests over programming projects for the same reason. My own feeling is that the Personal Computing approach is more effective, but what is more important than choosing sides is to note that the faculty should notice this contradiction themselves.

The positive side of this school’s approach, again, is that it *works*. A large number of students with diverse interests and abilities learn something about computers here, either to program them or to use them for word processing. The faculty has made an enormous effort in curriculum development, and continues to make an enormous effort in the actual teaching of students. All the teachers I interviewed are serious, thoughtful, and committed. This is the American, pragmatic, can-do philosophy at its best.

Educational Goals: John Dewey

As a background for the philosophical analysis of the goals discovered in educational computing, it will be helpful to review some of the goals discussed by philosophers of education in the context of education generally, not just educational computing. One important reference in the philosophical literature is John Dewey's 1938 essay *Experience and Education* (Dewey, 1963). My purpose is to show how Dewey's ideas can be applied specifically to computer-based education.

Dewey begins by suggesting that some proponents of progressive education have neglected to think about the positive philosophical basis of their work, relying instead on a simple negation of the traditional practices. Here is Dewey's own central positive idea: "I take it that the fundamental unity of the newer philosophy is found in the idea that there is an intimate and necessary relation between the processes of actual experience and education." (*ibid.*, p. 20.) He goes on to say that this doesn't mean that children had no experiences in traditional classrooms. On the contrary, people have experiences no matter what we plan for them. But some experiences are educative, and some are miseducative. Dewey examines all the usual questions about curriculum, discipline, and so on in the light of this idea of experience.

For example, he says this about authority:

When external control is rejected, the problem becomes that of finding the factors of control that are inherent within experience. When external authority is rejected, it does not follow that all authority should be rejected, but rather that there is need to search for a more effective source of authority. Because the older education imposed the knowledge, methods, and the rules of conduct of the mature person upon the young, it does not follow, except upon the basis of the extreme *Either-Or* philosophy, that the knowledge and skill of the mature person has no directive value for the experience of the immature. On the contrary, basing education upon personal experience may mean more multiplied and more intimate contacts between the mature and the immature than ever existed in the traditional school, and consequently more, rather than less, guidance by others. The problem, then, is: how these contacts can be established without violating the principle of learning through personal experience. The solution of this problem requires a well thought-out philosophy

of the social factors that operate in the constitution of individual experience. (*ibid.*, p. 21.)

Dewey returns later in the book to more detailed explanations of the issues raised in this paragraph, including the nature of experience and the social organization of learning. But I find two points that are already clearly applicable to the educational use of computers.

First, the skill of computer programming is one of the few practical, modern skills that can reasonably be the basis for something like an old-fashioned *apprenticeship*. This seems like one good way to solve Dewey's problem of combining guidance with the principle of learning through personal experience. Here is why programming is special in this respect: First, it's a skill with a quick payoff for young people, like juggling or playing the guitar, but unlike writing or trigonometry. Second, the particular skills that are part of computer programming are closely related to other parts of education because they are intellectual skills, reasoning skills, rather than (say) motor skills. Third, the computer itself imposes the standard of adequate performance; your program either does or doesn't work. This situation is in contrast to the usual one in which the standard is (or at least seems) arbitrarily imposed by the teacher. Fourth, there is still room for mature guidance, as the learner comes to see the value of ideas about programming style, for example, that go beyond a simple "does the program run?"

If the idea of programming as apprenticeship is taken seriously, it suggests some changes in the way programming is generally taught. The most important change is to see that not every child should be, or can be, a serious apprentice programmer. The level of commitment demanded of an apprentice is much greater than that which a "literacy" course can accommodate, let alone require. Another implication is that the apprentice programmer should work with serious, adult programming tools, not toy languages like BASIC.

The second major point I wish to point out in Dewey's paragraph is the intimate connection between "social factors" and curriculum. Dewey will have more to say about this later, as shall I, but the thing to keep in mind now is that questions like the conditions of access to the computers, access to teachers, and access to fellow students are related to what is being taught. The assignment of grades is another major social factor that affects the nature of the learner's experience.

What makes an experience educative or miseducative? Dewey discusses this question in terms of two principles: *continuity* and *interaction*. The continuity of experience is Dewey's name for the fact that each experience is part of the background for later ones.

[T]here is some kind of continuity in any case since every experience affects for better or worse the attitudes which help decide the quality of further experiences, by setting up certain preference and aversion, and making it easier or harder to act for this or that end. Moreover, every experience influences in some degree the objective conditions under which further experiences are had. For example, a child who learns to speak has a new facility and a new desire. But he has also widened the external conditions of subsequent learning. (*ibid.*, p. 37.)

In particular, each experience affects the learner's attitude about learning itself.

Perhaps the greatest of all pedagogical fallacies is the notion that a person learns only the particular thing he is studying at the time. Collateral learning in the way of formation of enduring attitudes, of likes and dislikes, may be and often is much more important than the spelling lesson or lesson in geography or history that is learned. For these attitudes are fundamentally what count in the future. The most important attitude that can be formed is that of desire to go on learning. If impetus in this direction is weakened instead of being intensified, something much more than mere lack of preparation takes place. The pupil is actually robbed of native capacities which would otherwise enable him to cope with the circumstances that he meets in the course of his life. (*ibid.*, p. 48.)

I think that this point speaks to the fundamental difference between the approach of Computer Managed Instruction, in which the computer takes charge of the child, and other approaches that let the computer serve the child. As Seymour Papert asks, "Should the computer program the child, or should the child program the computer?" Both active and passive learning were possible before computers. But the computer strongly intensifies the possibilities in both directions. One common educational use of computers, for drill and practice, does not encourage the learner to take the initiative in later experiences.

Dewey's second principle for evaluating experiences is interaction. "It assigns equal rights to both factors in experience—objective and internal conditions." (*ibid.*, p. 42.) That is, each experience is determined partly by what happens inside a person's head, and partly by what the person finds in the environment. It is the latter factor that can be affected directly by the teacher, but Dewey faults traditional education for ignoring the

internal factor. (He faults some progressive schools for thinking that *only* the internal is important.)

Failure to take into account adaptation to the needs and capacities of individuals was the source of the idea that certain subjects and certain methods are intrinsically cultural or intrinsically good for mental discipline. There is no such thing as educational value in the abstract. The notion that some subjects and methods and that acquaintance with certain facts and truths possess educational value in and of themselves is the reason why traditional education reduced the material of education so largely to a diet of predigested materials. According to this notion, it was enough to regulate the quantity and difficulty of the material provided, in a scheme of quantitative grading, from month to month and from year to year. (*ibid.*, p. 46.)

The computer has been widely promoted as a vehicle for “individualized instruction.” This phrase, although it can express a valuable idea, in practice often means drill programs with multiple skill levels. Since the slogan came from Dewey’s work in the first place, it is important to see what he meant by it. Here he explicitly rejects “regulat[ing] the quantity and difficulty of the material” as an adequate form of meeting the needs of individuals. Instead, Dewey’s approach requires a creative, human teacher who can react to each learner’s interests as well as his deficiencies.

What of the social organization of learning? Dewey proposes as a model the situation of a group of people playing a game. In order to carry on their joint project, the game, they willingly submit to rules that are an integral part of the game, not arbitrary impositions.

[I]n what are called the new schools, the primary source of social control resides in the very nature of the work done as a social enterprise in which all individuals have an opportunity to contribute and to which all feel a responsibility. Most children are naturally “sociable.” Isolation is even more irksome to them than to adults. A genuine community life has its ground in this natural sociability. But community life does not organize itself in an enduring way purely spontaneously. It requires thought and planning ahead. The educator is responsible for a knowledge of individuals and for a knowledge of subject-matter that will enable activities to be selected which lend themselves to social organization, an organization in which all individuals have an opportunity to contribute something, and in which the activities in which all participate are the chief carrier of control. (*ibid.*, p. 56.)

Perhaps the most obvious implication for computer education in what Dewey says is that he would not favor the style of computer use in which each student sits in an isolated cubby, interacting only with the computer. A computer center can be designed to encourage a much more social style of computing. One aspect of this design is architectural: computer users should be able to see each other's screens, and get together for impromptu group discussions of technical problems. Another aspect is organizational: learners should be involved in decisions regarding resource allocation and so on. A third aspect is technical: users should be able to share files, through the use of a timeshared machine or a network with a file server. Electronic mail can contribute to making the computer a social focus.

Dewey is very interested in the relationship between education and the broader questions of social organization. In the book *Democracy and Education* (Dewey, 1916) he pursues the idea that all experiences are educative, and that formal education carried out in schools is just one particular form of experience. For Dewey, education is valuable as an end in itself, and so he justifies a democratic form of society on the grounds that democracy is more educative than despotism, by contrast with the more conventional thinking that would start with democracy as a primary good and ask how education can serve democracy.

If we take... the kind of family life which illustrates the standard, we find that there are material, intellectual, æsthetic interests in which all participate and that the progress of one member has worth for the experience of other members—it is readily communicable—and that the family is not an isolated whole, but enters intimately into relationships with business groups, with schools... In short, there are many interests consciously communicated and shared; and there are varied and free points of contact with other modes of association. (*ibid.*, p. 83.)

Dewey uses the family as a metaphor for a well-structured society. It is well-structured because it is educative: it provides its members with experiences that will lead to growth. Each person supports the growth of each other person. By participating in the affairs of the world, the family makes a wide range of experiences and viewpoints available to its members. Despotism is anti-educational because the common interests between the rulers and the ruled are not educative; each group is limited in the kinds of experiences made available to it.

It is not true that there is no common interest in such an organization between governed and governors. The authorities in command must make some appeal to the native activities of the subjects, must call some of their powers into play... It may be said, however, that the activities appealed to are themselves unworthy and degrading—that such a government calls into functioning activity simply capacity for fear... The real difficulty is that the appeal to fear is *isolated*. In evoking dread and hope of specific tangible reward—say comfort and ease—many other capacities are left untouched. Or rather, they are affected, but in such a way as to pervert them. Instead of operating on their own account they are reduced to mere servants of attaining pleasure and avoiding pain. (*ibid.*, p. 84.)

Similarly, a social group that is organized in opposition to other groups, like a criminal gang, is miseducative because it limits the experience of its members.

Dewey's analysis of democracy as an instrumental good in the service of education can be applied in the school as well as in the larger society. Often the question of student self-government is discussed as a matter of the rights of children in conflict with the interest of adults, but Dewey would reject that view. Instead he would suggest that student autonomy should be supported by teachers as part of their educative work. This does not mean that teachers should never tell students what to do; as in the analogy of the family, student autonomy should be encouraged to the extent that the students' capabilities allow. Another part of the family analogy is that teachers and students are part of one community in which each member's goals are important, teachers as well as students, and the adult members of the community are still learning and growing through their own work.

Dewey's point about the relationship between one small group of people and other groups in a democratic, pluralistic society has a direct relevance to computer education. Dewey does not approve of cliques, but neither does he want a uniform, bland social group. His ideal society allows for smaller groupings, in which shared enterprises make for mutual support within the group, but the groups must look outward as well as inward. Computer enthusiasts form a subgroup within a school, but in most schools this group is either discouraged because of the intensity of its interest (as at school E) or isolated as "computer nerds" relative to the larger community (as at school C). Dewey would encourage the development of a community of computer experts, as a rich source of educative experience, but he would also push the group to feel themselves a part of the school as a whole, and

encourage the rest of the school to accept them both as a special group and as part of the larger community.

Dewey often repeats his fundamental idea that education is experience, and that all experience is educative, not just formal schooling. Why, then, do we bother to set up the institution of the school? Dewey rejects the idea that there is a fundamental difference between learning from experience in ordinary life and learning by the direct transmission of ideas by a teacher in school. “[T]he only way in which adults consciously control the kind of education which the immature get is by controlling the environment in which they act, and hence think and feel. We never educate directly, but indirectly by means of the environment.” (*ibid.*, p. 18.) The importance of the school is that it is a *planned* environment, in which the elements are chosen specifically for educational purposes. Dewey lists three special characteristics of the school environment. First, it is *simplified*. “A complex civilization is too complex to be assimilated *in toto*. It has to be broken up into portions...” (*ibid.*, p. 20.) Second, it is *purified*. “It is the business of the school environment to eliminate, so far as possible, the unworthy features of the existing environment from influence upon mental habitudes.” Third, “it is the office of the school environment to balance the various elements in the social environment, and to see to it that each individual gets an opportunity to escape from the limitations of the social group in which he was born, and to come into living contact with a broader environment.”

School, then, should be a carefully selected slice of life, a controlled environment in which young people will have educative experience. But in fact the flavor of most schooling is not like this; instead, the experiences in a school are mostly verbal, not physical; mostly passive, not active. Dewey explains this phenomenon as having a good part and a bad part.

Intentional education signifies, as we have already seen, a specially selected environment, the selection being made on the basis of materials and method specifically promoting growth in the desired direction. Since language represents the physical conditions that have been subjected to the maximum transformation in the interests of social life—physical things which have lost their original quality in becoming social tools—it is appropriate that language should play a large part compared with other appliances. By it we are led to share vicariously in past human experience, thus widening and enriching the experience of the present. We are enabled, symbolically

and imaginatively, to anticipate situations. In countless ways, language condenses meanings that record social outcomes and presage social outlooks. So significant is it of a liberal share in what is worth while in life that unlettered and uneducated have become almost synonymous.

The emphasis in school upon this particular tool has, however, its dangers—dangers which are not theoretical but exhibited in practice. Why is it, in spite of the fact that teaching by pouring in, learning by a passive absorption, are universally condemned, that they are still so intrenched in practice? That education is not an affair of “telling” and being told, but an active and constructive process, is a principle almost as generally violated in practice as conceded in theory. Is not this deplorable situation due to the fact that the doctrine is itself merely told? It is preached; it is lectured; it is written about. But its enactment into practice requires that the school environment be equipped with agencies for doing, with tools and physical materials, to an extent rarely attained. It requires that methods of instruction and administration be modified to allow and to secure direct and continuous occupations with things. Not that the use of language as an educational resource should lessen; but that its use should be more vital and fruitful by having its normal connection with shared activities. “These things ought ye to have done, and not left the others undone.” And for the school “these things” mean equipment with the instrumentalities of coöperative or joint activity.

For when the schools depart from the educational conditions effective in the out-of-school environment, they necessarily substitute a bookish, a pseudo-intellectual spirit for a social spirit. Children doubtless go to school to learn, but it has yet to be proved that learning occurs most adequately when it is made a separate conscious business. When treating it as a business of this sort tends to preclude the social sense which comes from sharing in an activity of common concern and value, the effort at isolated intellectual learning contradicts its own aim. We may secure motor activity and sensory excitation by keeping an individual by himself, but we cannot thereby get him to understand the meaning which things have in the life of which he is a part. We may secure technical specialized ability in algebra, Latin, or botany, but not the kind of intelligence which directs ability to useful ends. Only by engaging in a joint activity, where one person’s use of material and tools is consciously referred to the use other persons are making of their capacities and appliances, is a social direction of disposition attained. (*ibid.*, p. 38.)

The central point of this passage is the one about learning by doing, rather than by passive listening. But at the same time, Dewey is making the case for learning as a *social* activity. For him, working with other people is not just one more piece of the curriculum, like

“citizenship,” but rather an essential part of learning to act purposefully in any subject area. In particular, the problem solving that computer educators often list as a goal is only meaningful in the context of “an activity of common concern and value.” Of course, Dewey would see the computer itself as a manipulable object, a tool, rather than as a high-tech filmstrip, the repository of information to be absorbed passively by students.

What educators today call problem solving is what Dewey called thinking, or *reflection on experience*. For Dewey, thinking is not one goal among many; it must be a part of every school activity.

[T]here is not adequate theoretical recognition that all which the school can or need do for pupils, so far as their *minds* are concerned (that is, leaving out certain specialized muscular abilities), is to develop their ability to think. The parceling out of instruction among various ends such as acquisition of skill (in reading, spelling, writing, drawing, reciting); acquiring information (in history and geography), *and* training of thinking is a measure of the ineffective way in which we accomplish all three. (*ibid.*, p. 152.)

Thinking starts with experience “which is not either routine or capricious—something, in other words, presenting what is new (and hence uncertain or problematic) and yet sufficiently connected with existing habits to call out an effective response.” (*ibid.*, p. 154.)

Dewey goes on to describe conditions that must be met for such a problematic experience to lead to reflective thought. In summary, the student must have access to the necessary facts and other tools to attack the problem, and it must be one that gives rise to new ideas, approaches, guesses. Much of what happens in any school is the posing of problems for students, but Dewey says that it is only on the surface that these problems seem to promote thinking.

But it is indispensable to discriminate between genuine and simulated or mock problems. The following questions may aid in making such discrimination. (*a*) Is there anything but a problem? Does the question naturally suggest itself within some situation or personal experience? Or is it an aloof thing, a problem only for the purposes of conveying instruction in some school topic? Is it the sort of trying that would arouse observation and engage experimentation outside of school? (*b*) Is it the pupil’s own problem, or is it the teacher’s or the textbook’s problem, made a problem for the pupil only because he cannot get the required mark or be promoted or win the teacher’s

approval, unless he deals with it? (*ibid.*, p. 155.)

In the elementary schools, where computer programming is still seen largely as an “enrichment” activity, students are often given the freedom to work on their own problems. But in the high schools, computers have become serious business, part of the “real” curriculum, and so there are textbooks with problem sets at the end of each chapter. Dewey would recommend that students be left with the opportunity to use the computer, an unprecedentedly flexible tool, for their own ends. High school students who have had such an opportunity have written programs ranging from a simple repetitive printing of hexagonal graph paper for strategy games up to a full-scale display text editor.

Educational Goals: Martin Buber

Dewey's theory of learning is that it is grounded in personal experience. Experience of what? Dewey's view is fundamentally that of the scientist: one experiences the objective world, a world that is there waiting before the learner experiences it. The learner's interaction with that world is important, but the learner does not establish the world. Instead, it is the learner's self that is established, developed, through the process of education.

For Martin Buber (1878–1965), the substance of learning lies neither in the objects of the curriculum (as for traditional education) nor in the learner's developing mental processes (as in some forms of progressive education), but rather in the dialogue between the learner and the world. The job of the teacher is to represent the world to the student, and to decide which parts of the world's complexity should be represented for the student in each immediate situation. Buber's emphasis is on the *human* relationship between a particular teacher and a particular student, rather than on what is taught or on the teacher as a specialist in the techniques of teaching.

Like Dewey, Buber objects to the dichotomy between the old education, based on dead authority, and the new education, based on the creative freedom of the learner. He objects to the phrase “development of creative powers” as a description of the task of education. He applauds the developers of that phrase for noticing an important, previously unrecognized precondition for education:

I mean the existence of an autonomous instinct, which cannot be derived from others, whose appropriate name seems to me to be the “originator instinct.” Man, the child of man, wants to make things. (Buber, 1955, p. 85.)

This instinct is not an urge to busyness, nor is it libido, the will to power, or some other primary drive in disguise. It is partly this instinct that makes education possible, and the instinct can only thrive in an atmosphere of freedom and spontaneity. “But the release of powers should not be any more than a *presupposition* of education.” (*ibid*, p. 88.) Education is necessary precisely because the originator instinct alone is inadequate for the building of human character.

There are two forms, indispensable for the building of true human life, to which the originaive instinct, left to itself, does not lead and cannot lead: to sharing in an undertaking and to entering into mutuality. (*ibid.*, p. 87.)

An undertaking is a job done by a group. It involves making something, like the instinctive origination, but is different in that it involves “a community of work with other men.” It is through such community that we escape from painful solitariness. We need community because we have an “instinct for communion” that is as important as the instinct for origination.

Buber is making a strong demand here. To have a *community* of work, it is not enough that there be work that is shared. The work must make sense, and the sharing of the work must make sense, in terms of each person’s whole self, and his understanding of the others’ selves. For example, the assignment of computer programming problems to small groups rather than individual students is a good step, from Buber’s perspective, but I think he would say that it is ruined by urging the students to select partners who are *not* their friends—in other words, to make this working group a walled-off entity separate from the real life of its members.

Why is the human relationship between teacher and student so important? Buber gives the example of an art class. A traditional teacher would begin by dogmatically telling students the official rules for beautiful art, and require the students to copy old masterpieces. (Perhaps the teaching of structured programming as a dogmatic system of rules fits into the same category.) A “free” teacher would begin by giving the students a model to draw, but saying nothing about rules of technique. The traditional approach is doomed from the start, because young spirits must react to dogmatic imposition either by rebellion, in which case they learn nothing, or by apathetic submission, in which case they learn the rules at the expense of their strength of character. The progressive approach *begins* by encouraging exploration; so far, so good. But if the progressive teacher *stops* with this offer of freedom, no education takes place. The teacher must, in time, make her own (and, through herself, the world’s) critical values available to the student. But the student can only accept such criticism if she first trusts the teacher. This trust can only grow out of a true human relationship.

It is possible to find faint echos of Buber's position in the literature of technical education. The teacher's guides for the Madison Project mathematics curriculum talk about the need for an atmosphere of trust. The research reports from the MIT Logo Laboratory talk about encouraging exploration and even feeling good about the mistakes in a computer program. But in both these examples, the atmosphere of trust is based on relatively mechanical strategies: grading policies, for example, that do not penalize errors. The interaction between teacher and student, in these models, is still that of a professional helper dealing with a small aspect of the student's life. There is no committed interaction of whole people, at least not as a necessity for education to be possible at all.

Buber recognizes the emergence of education, as a profession, as a necessary part of modern technological society. He agrees with Dewey that the whole world, the whole of a person's experience, educates. What is new, in our time, is the idea of *deliberate* education.

The world, that is the whole environment, nature and society, "educates" the human being: it draws out his powers, and makes him grasp and penetrate its objections. What we term education, conscious and willed, means *a selection by man of the effective world*: it means to give decisive effective power to a selection of the world which is concentrated and manifested in the educator. The relation in education is lifted out of the purposelessly streaming education by all things, and is marked off as purpose. In this way, through the educator, the world for the first time becomes the true subject of its effect. (*ibid.*, p. 89.)

Buber contrasts this modern sense of education with the older model of apprenticeship.

There was a time, there were times, where there neither was nor needed to be any specific calling of educator or teacher. There was a master, a philosopher or a coppersmith, whose journeymen and apprentices lived with him and learned, by being allowed to share in it, what he had to teach them of his handwork or brainwork. But they also learned, without either their or his being concerned with it, they learned, without noticing that they did, the mystery of personal life: they received the spirit. Such a thing must still happen to some extent, where spirit and person exist, but it is expelled to the sphere of spirituality, of personality, and has become exceptional, it happens only "on the heights." Education as a purpose is bound to be summoned. We can as little return to the state of affairs that existed before there were schools as to that which existed before, say, technical science. But we can and must enter into the completeness of

its growth to reality, into the perfect humanization of its reality. Our way is composed of losses that secretly become gains. Education has lost the paradise of pure instinctiveness and now consciously serves at the plough for the bread of life. It has been transformed; only in this transformation has it become visible. (*ibid.*, p. 90.)

In the old days, an apprentice lived in his master's house and shared not only his work but also his meals, his recreation, and all the other parts of his daily life. In this setting, a full human relationship grew between them automatically, as between parent and child. The modern teacher sees his students only in the formal setting of the classroom; therefore, he must do by deliberate action what the master did automatically. If the teacher allows the classroom setting to lead to a formal, "professional" relationship instead of a full human one, the student will see his teaching as interference and will resist it. This is what Buber means by the need to enter into mutuality.

Yet the master remains the model for the teacher. For if the educator of our day has to act consciously he must nevertheless do it "as though he did not." That raising of the finger [in criticism], that questioning glance, are his genuine doing. Through him the selection of the effective world reaches the pupil. He fails the recipient when he presents this selection to him with a gesture of interference... Interference divides the soul in his care into an obedient part and a rebellious part. But a hidden influence proceeding from his integrity has an integrating force. (*ibid.*, p. 90.)

When I discussed apprenticeship in connection with Dewey's idea of learning through experience, it was with the emphasis on the relationship between the apprentice and the *work*: young people need real tasks rather than abstract exercises. Buber would agree with that, but his real emphasis is on the relationship between the apprentice and the *master*. This difference is a valid insight, but at the same time, Buber would say that the distinction is artificial: *there is no real work* without real human community.

Buber continues his critique of "free" education by discussing the nature of freedom. He disagrees with those who see freedom as the opposite of compulsion, and who therefore make freedom an end in itself. For Buber, the opposite of compulsion is communion—that is, the choice is not between life in a group and life alone, but between forced group life and voluntarily chosen group life. No man is an island, and individual freedom as a goal in itself is empty.

The release of powers can be only a presupposition of education, nothing more. Put more generally, it is the nature of freedom to provide the place, but not the foundation as well, on which true life is raised. That is true both of inner, “moral” freedom and of outer freedom (which consists in not being hindered or limited)...

There is a tendency to understand this freedom... as at the opposite pole from compulsion, from being under a compulsion. But at the opposite pole from compulsion there stands not freedom but communion. Compulsion is a negative reality; communion is the positive reality; freedom is a possibility, possibility regained. At the opposite pole of being compelled by destiny or nature or men there does not stand being free of destiny or nature or men but to commune and to covenant with them. To do this, it is true that one must first have become independent; but this independence is a foot-bridge, not a dwelling-place. Freedom is the vibrating needle, the fruitful zero. Compulsion in education means disunion, it means humiliation and rebelliousness. Communion in education is just communion, it means being opened up and drawn in. Freedom in education is the possibility of communion; it cannot be dispensed with and it cannot be made use of in itself; without it nothing succeeds, but neither does anything succeed by means of it: it is the run before the jump, the tuning of the violin, the confirmation of that primal and mighty potentiality which it cannot even begin to actualize...

It is easy to understand that in a time when the deterioration of all traditional bonds has made their legitimacy questionable, the tendency to freedom is exalted, the springboard is treated as the goal and a functional good as substantial good... To become free of a bond is destiny; one carries that like a cross, not like a cockade. Let us realize the true meaning of being free of a bond: it means that a quite personal responsibility takes the place of one shared with many generations. Life lived in freedom is personal responsibility or it is a pathetic farce. (pp. 90–92.)

What is the nature of the proper relationship between teacher and student? Buber rejects three possibilities found in other philosophies of education. The relationship cannot be, as in much modern thinking, one of professional detachment. The teacher must care about the student not only as one of many clients, but as an individual human person. However, Buber rejects the characterization of the “old” education as reflecting the will to power, and of the “new” education as representing Eros. The old, authoritarian education, when honestly pursued, is not the expression of the teacher’s will to power, but rather “assured values which were strong in tradition... the historical world, the past.” (*ibid.*, p.

93.) Of course this tradition can be distorted by the will to power of a particular teacher, but that is a failing in the teacher and not the proper result of the traditional philosophy. Similarly, the “new” educator, though he may be inspired by Eros, cannot carry out his job in an erotic spirit. “Eros is choice, choice made from an inclination. This is precisely what education is not. The man who is loving in Eros chooses the beloved, the modern educator finds his pupil there before him.” (*ibid.*, p. 94.)

If the human relationship between teacher and student is not the expression of the teacher’s instinctive drive for power or Eros, those drives are not irrelevant either. They can serve as the basis for the experience that is at the heart of Buber’s idea of education: the relationship of *inclusion*. Inclusion is one person’s experience of another person’s being.

There is an elemental experience which shatters at least the assurance of the erotic as well as the cratetic man, but sometimes does more, forcing its way at white-heat into the heart of the instinct and remoulding it. A reversal of the single instinct takes place, which does not eliminate it but reverses its system of direction. Such a reversal can be effected by the elemental experience with which the real process of education begins and on which it is based. I call it experiencing the other side. (*ibid.*, p. 96.)

Buber gives an example of a man, driven by the will to power, striking another. “Then let us assume that the striker suddenly receives in his soul the blow which he strikes: the same blow, that he receives it as the other who remains still.” He has experienced the other side, and his impulse to power must be reversed. The other person has become a person for him, rather than merely the object of his own needs. Similarly, a lover “feels the contact from two sides—with the palm of his hand still, and also with the woman’s skin.” Then his relationship with the woman is no longer purely erotic; he must recognize her as a person also.

Buber is going to take this idea of experiencing the other side, of inclusion, and make it the basis of the teaching relationship. He does not mean that the teacher must begin by striking or caressing the student, but his use of these strong experiences as the basis for understanding inclusion is not irrelevant to education; our strong human drives underlie any experience that can genuinely bridge the gap between two people.

It would be wrong to identify what is meant here with the familiar but

not very significant term “empathy.” Empathy means, if anything, to glide with one’s own feeling into the dynamic structure of an object, a pillar or a crystal or the branch of a tree, or even of an animal or a man, and as it were to trace it from within, understanding the formation and motoriality of the object with the perception of one’s own muscles; it means to “transpose” oneself over there and in there. Thus it means the exclusion of one’s own concreteness, the extinguishing of the actual situation of life, the absorption in pure aestheticism of the reality in which one participates. Inclusion is the opposite of this. (*ibid.*, p. 97.)

The special nature of the educational relationship comes from the fact that it is not symmetrical. The teacher must include the student so that his teaching—his selection of the world to present to the student—will be based on his sure understanding of the individual student’s current, situated needs. Otherwise the teaching “may degenerate into arbitrariness.” But the student does not yet have the capacity to include the teacher within himself. “The educator stands at both ends of the common situation, the pupil only at one end.” (*ibid.*, p. 100.) When the student becomes able to include the teacher, he is no longer a student, but simply a friend. This does not mean that the student does not participate in the relationship at all. The relationship is very important to the student, who must be aware of it in order to be able to accept the critical function of the teacher and still feel unthreatened. Not only does the student trust the teacher, the student *trusts in the world* because the teacher is present for him.

What should the teacher’s goal be, in the educational relationship? “Education worthy of the name is essentially education of character.” (*ibid.*, p. 104.) A teacher who is concerned with the student as a whole person cannot say, “My job is to teach you how to program a computer; what you do with that is your business.” Buber says that we cannot educate the *personality*, the unique nature of each person, but we can educate the *character*, the manifestation of that unique person in “the sequence of his actions and attitudes.”

Even character education is very difficult. It cannot be done by giving lectures in ethics. Such lecturing may be accepted as representing what the adults think, but it can have no effect on a student’s character. “I try to explain that it is wicked to bully the weak, and at once I see a suppressed smile on the lips of the strong.” (*ibid.*, p. 105.) In fact, students will actively resist any effort to educate their characters; those who have the

most independent characters will resist most strongly. Nor can the difficulty be solved by teaching ethical precepts surreptitiously.

Does it follow that one should keep silent about one's intention of educating character, and act by ruse and subterfuge? No; I have just said that the difficulty lies deeper. It is not enough to see that the education of character is not introduced into a lesson in class; neither may one conceal it in cleverly arranged intervals. Education cannot tolerate such politic action. Even if the pupil does not notice the hidden motive it will have its negative effect on the actions of the teacher himself by depriving him of the directness which is his strength. Only in his whole being, in all his spontaneity can the educator truly affect the whole being of his pupil. For educating characters you do not need a moral genius, but you do need a man who is wholly alive and able to communicate himself directly to his fellow beings. His aliveness streams out to them and affects them most strongly and purely when he has no thought of affecting them. (*ibid.*, p. 105.)

In computer education, Buber is saying that we cannot teach "computer ethics" through a class lecture about why not to copy software. This moral imposition will not be effective as an abstract lecture, nor even as a practical rule in our computer centers. Students may obey the rule through fear of punishment, but the rule will not become part of their own being by imposition.

[I]n this realm of the education of character, of wholeness, there is only *one* access to the pupil: his *confidence*. For the adolescent who is frightened and disappointed by an unreliable world, confidence means the liberating insight that there is human truth, the truth of human existence. When the pupil's confidence has been won, his resistance against being educated gives way to a singular happening: he accepts the educator as a person. He feels he may trust this man, that this man is not making a business out of him, but is taking part in his life, accepting him before desiring to influence him. And so he learns to *ask*. (*ibid.*, p. 106.)

Buber speaks in poetic language, and perhaps he leaves the impression that his ideas are beautiful but of no practical significance. Here is a more down-to-Earth example of what I think he means. Most high school teachers of computer programming are not, themselves, expert programmers; the experts can earn more money for less effort by working as programmers in industry. When I taught programming in a rich, suburban school, several

students noticed that I was expert enough to be able to do this. “What are you doing here,” they asked. Many of their own parents worked at jobs they found uninspiring in order to pay for the suburban homes, the large lawns, and the college tuitions of their children. I could not have affected the thinking of these students by giving a lecture on the theme that there is more to life than money. But some of my students came seriously to ask me, of their own will, to explain the feelings that brought me to choose a relatively low-paying job when I could have chosen otherwise. It is important that I did not deliberately choose to display an example of valuing a calling over money *in order* to educate my students’ characters. The example arose out of a true, ingenuous choice in my own wholeness as a person. Only in that context would it meet Buber’s criterion.

If the teacher takes the student seriously, and takes part in his whole life as a whole person, then the teacher cannot pretend to ethical neutrality. His moral teaching cannot be values clarification: “He must use his whole insight wholeheartedly.” But at the same time the teacher must accept the possibility that a student may not agree with his values: “Conflicts too, if only they are decided in a healthy atmosphere, have an educational value.” (*ibid.*, p. 107.) The precondition for this healthy atmosphere is mutual acceptance and trust.

Buber goes on to say that the education of character is particularly difficult in our era because we have lost the sense of absolute, universal values. As a religious man, he deplores ethical relativism. But he does not feel that this dispute can be won by argument.

[I]t is senseless to want to prove by any kind of argument that nevertheless the denied absoluteness of norms exists. That would be to assume that the denial is the result of reflection, and is open to argument, that is, to material for renewed reflection. But the denial is due to the disposition of a dominant human type of our age. We are justified in regarding this disposition as a sickness of the human race... It is an idle undertaking to call out, to a mankind that has grown blind to eternity: “Look! the eternal values!” (*ibid.*, p. 110.)

Instead, says Buber, “One has to begin by pointing to that sphere where man himself, in the hours of utter solitude, occasionally becomes aware of the disease through sudden pain: by pointing to the relation of the individual to his own self.” This both is and is not similar to the idea of providing psychological counseling in school. The similarity is that the school, in Buber’s view, must not draw a line between the student’s intellectual life and

his emotional or spiritual life; if he is in pain, that is a matter for education. The difference is that Buber's goal is not to soothe or hide the pain, but to expose it, to heighten it, as a way of awakening the individual spirit.

Buber quotes Dewey's definition of character as "the interpenetration of habits." Buber disagrees:

[T]his concept can be no adequate basis for the construction of a genuine education of character. Not that the educator could dispense with employing useful maxims or furthering good habits. But in moments that come perhaps only seldom, a feeling of blessed achievement links him to the explorer, the inventor, the artist, a feeling of sharing in the revelation of what is hidden. In such moments he finds himself in a sphere very different from that of maxims and habits. (*ibid.*, p. 112.)

The teacher's goal is to educate "the great character." Not every student will be a great character; the teacher cannot choose his students but must teach everyone in his class. For many students, he may accomplish no more than establishing a habit of discipline and law. But the great character does not act out of habit, but "from the whole of his substance. That is, it is peculiar to him to react in accordance with the uniqueness of every situation which challenges him as an active person." (*ibid.*, p. 113.) This does not mean that the great character refuses to recognize rules. Buber discusses how a social norm that other people learn by rote can be revealed to a great character as part of his own personhood.

Finally, Buber asks whether it is reasonable for the teacher to seek the great character in his work, when in fact most of his students will not develop such character.

Of course, it may be asked whether the educator should really start "from above," whether, in fixing his goal, the hope of finding a great character, who is bound to be the exception, should be his starting-point; for in his methods of educating character he will always have to take into consideration the others, the many. To this I reply that the educator would not have the right to do so if a method inapplicable to these others were to result. In fact, however, his very insight into the structure of a great character helps him to find the way by which alone... he can begin to influence also the victims of the collective Moloch... (*ibid.*, p. 115.)

In this important remark, Buber is arguing against a tracked education. "The great character" is not the same thing as intelligence, but they are not unconnected; Buber would

have us teach every class as though it were a Gifted and Talented class. The methods that inspire the best work from gifted children also inspire anyone: freedom, open-ended projects, intrinsic motivation.

Buber's most famous idea is the profound difference between the *I-Thou* relationship of mutuality appropriate to human beings and the *I-It* relationship between a human being and an inanimate tool or object of manipulation. The computer threatens to blur this difference because it is an object that can talk back. Its conversational ability can mimic, to an extent, an interaction with a person, and yet the computer is fundamentally mechanical. I think Buber would see two dangers in this characteristic of computers. First, some people may learn to interact with computers as their primary or only "social" life, accepting a simulation of true social interaction because the computer is a more predictable, less risky companion than another human being. This is a picture of the stereotype "computer nerd." Second, if people learn to use the computer as a metaphor through which to understand human beings, as cognitive psychologists do, this mechanistic metaphor may allow people to treat other people as appropriate objects for an I-It relationship, using them manipulatively instead of experiencing them in reciprocal inclusion. Sociologist Sherry Turkle has studied the ways in which the computer metaphor has actually affected computer users' image of themselves and of other people (Turkle, 1984). Her conclusions are ambiguous; she finds both helpful and harmful potential in the computer metaphor. If there is a danger in a computer metaphor for humanity, the implication for education is that teachers in computer-based classrooms must be especially concerned with presenting themselves as full human beings, to be sure that students are aware of the differences between interaction with an object and mutuality with another subject like themselves.

Educational Goals: Frank Oppenheimer

Philosophers are valuable because they stand back from the details of a specific curriculum area, a specific locale, or a specific pragmatic need; instead they teach us about more general perspectives on the educational process in general. But it may also be helpful to look at the thinking about goals by a successful educational practitioner in an area different from the high school computer center. Frank Oppenheimer (1912–1985) was the founder of the Exploratorium, a world-famous science museum that has set a new standard for the meaning of “interactive” in describing exhibits. After his death in February, 1985, the museum published a special issue of *The Exploratorium* magazine containing some of his essays about education and science.

In the article “Teaching and Learning” (Oppenheimer, 1985), taken from a talk given to a high school Parent-Teacher Association in 1957, Oppenheimer lists some of his goals as a science teacher.

I believe that the major reason that I want to teach is to communicate my appreciation of and skill in science to the children. This motive is a simple motive, not very different from someone who exclaims to a companion while driving down the highway, “Look! There go three elk.” ...

I suppose therefore that the first thing I try to do as a teacher is to get my student to understand so clearly some phenomena or device, such as the twinkling of a star or the ring of an electric bell, that they realize that understanding, like eating or making a basket during a ball game, is satisfying and fun. If I can succeed in making understanding seem like fun then I believe that the student will want to understand many things, that is, he will become curious. (*ibid.*, p. 8.)

This goal is analogous, in certain ways, to that of “computer literacy.” One point of similarity is that the goal is addressed to every student, not just to a few who will become professional scientists. Another is that there is no *specific* skill or knowledge necessarily associated with the goal; it can have different details for different people. There are also some differences, however. One is that there is no fear behind it, no hint that science appreciation will be necessary to get a job in the future society. There is no shame associated with failure: “uncurious about science” does not sound as ominous as “computer illiterate.”

Another is that the goal is chiefly addressed to the student's attitude, rather than her knowledge. Oppenheimer does not say "the goal is that the student should understand the twinkling of a star." Instead he tries to promote that understanding as a *means* to teach that understanding is "satisfying and fun."

I believe that another motive I have as a teacher is to prepare the students for further learning. Although I know that in reality many of the students may not learn one more thing about science than they find in my biology course, I find that I teach everyone as though they were going to continue learning the subject... Thus, as I am teaching, I find that I have in the back of my mind what the content of the next course in the subject will be. I want this next course to be easy for them, but I do not want it to be entirely "old stuff." I find this distinction very hard to draw... because a certain amount of fuzziness and puzzlement is probably good for the more advanced students. (*ibid.*, p. 8.)

What would this goal mean in the field of computer education? To prepare students for college-level computer science is not the same as to teach it to them. That is, the Advanced Placement course, while perhaps worthy for other reasons, is not the answer to this issue. The question, rather, is what should be the content of a beginning course. I think one part of the answer should surely be that a beginning course and an advanced course should not work at cross purposes. This is the objection I raised to the Computer Discovery course at school A, but it is also an objection to the teaching of BASIC to beginners at a school that really believes in Pascal.

Finally, I try in my teaching to give the students a sense of power to actually do something: to teach them for example, to get numerical answers, bend a piece of glass, recognize a *Spirogyra* when they see one, or solve an unfamiliar problem. I think that I find this last objective hardest to fulfill.

Yet most people, and adolescents especially, are eager to become proficient in as many things as they can. In fact a frequent interpretation of education is limited solely to the belief that students should learn how to do things. One of the important aspects of sports is that they enable a substantial number of kids to become really good at something: catching a pass or pitching a ball or working with a group. I believe that a great many students enjoy and are helped by algebra because of the delightful opportunity for proficiency it affords in solving equations. Shop work, sewing, writing book reviews, typing, language courses, band and art are all important, not only because of

the useful skills they teach, but because in each one a different group of students may find that they can do something well. (*ibid.*, p. 8.)

Perhaps the most important comment to make about this third goal is that it *is* third. Oppenheimer does not make skill development the central issue in education. Also, he does not say that any *particular* skill is crucial. It is the experience of competence in itself that is important. Different students may do well at different fields; there is no suggestion that every student must be “sports literate” or “algebra literate.”

Oppenheimer goes on to say that, in fact, few of his students end up proficient in the science skills he teaches. He lists three problems that interfere with success in this goal. One is “the enormous variation in the initial ability of the students.” A second is lack of time and money. The third is specifically related to the goal of problem solving:

And finally, I do not really know how to teach children to work out problems for themselves. I can give them a problem to work out and say “think.” But this procedure is about as effective as saying “wiggle your ears.” I cannot tell them how to think, which nerves to use. Alternatively, I can have them follow me step by step as I reason something out, trying to let them get one step ahead of me. But usually the effect is about the same as if I had shown a small child how to saw a piece of wood by standing behind him and making his arms move the saw. The child would know what is required of him and what the motions are, but he still could not do it himself. In short, how does one best teach the most satisfying of all proficiencies: the ability to fashion a new idea. (*ibid.*, p. 9.)

This paragraph was written in 1957, before problem solving became fashionable. In it, Oppenheimer points out that setting problems is not enough; building a structured environment full of problem-solving tools is not enough; working through problems and solutions with students is not enough. Today some teachers think that computer programming automatically teaches the knack, but this is not so. Computer programming is a good arena in which to explore problem solving, but a theory of how people actually do solve problems is still needed.

In another article in the same issue, “Adult Play,” Oppenheimer adds another goal.

The problem of talking about adult play becomes difficult because there are some people who are never playful in their work or in their studying. I once asked Bob Karplus, a professor at U.C. Berkeley, a question that I had asked many people without getting a very

satisfactory answer. I asked, “Are there any things which a young person must learn before it is too late to learn them?” There has been much emphasis on how *early* a child can learn to write or to spell or to add, but my question seems not to have attracted much attention. There may very well not be anything which one has to learn before it’s too late to do so. But Bob answered, “Maybe people have to learn how to play before it’s too late.” (*ibid.*, p. 14.)

Play is important to Oppenheimer because it is the source of creativity in general and of scientific research in particular. Of course the computer is an excellent plaything. One likely consequence of this goal is that Oppenheimer would want to encourage the use of computer games. But playing does not mean only playing games; computer programming, for example, can be approached as play. Many things make it hard to play in a school. Obvious ones are grades and exams. A less obvious one is uniform programming assignments for an entire class. In the elementary schools, a widespread practice is to teach students the fundamental rules of Logo turtle graphics programming, and then let each child explore individually. In the high schools in this study, however, all assigned specific programming projects as homework or classwork for a particular day. Another obstacle to a playful spirit, less directly related to the school as an institution, is the top-down structured philosophy of the Pascal language. A Pascal programmer is supposed to start with a well-defined problem, then work out on paper a well-defined solution to it, and only then start writing the actual computer instructions. This may be a good strategy in a commercial programming environment, but it is not playful.

In a third article, “Everyone Is You...Or Me,” Oppenheimer discusses the goals that are involved in designing an exhibit for the Exploratorium.

Consider our audience contemplating a tree. Science museums all describe themselves as having interactive, involving, hands-on exhibits. But they misunderstand the implications of the terms they use. A tree has no pushbuttons, no cranks, no manipulative parts; but there are a lot of ways of interacting with it. One can look at it, lie under it, climb and feel it. One can watch the leaf buds unfold... One can draw or photograph the tree, carve initials on it, chop it down, or just stand and watch the sunlight diffract around the edges of the leaves. One can even learn its name.

Our exhibits do not have quite this versatility. For one thing, we do not want people to chop them down... More importantly, however,

we conceive most of our exhibit pieces as props to link a pedagogical chain; frequently the links are common to several different chains. Thus, the *Relative Motion Swing*, which has a swinging table beneath a pendulum of the same period, can be used in many contexts. One can use it in talking about vectors, about polarized light, about Lissajous figures... For each of these topics this exhibit is but one link shared by several other chains of exhibits, which may intersect at other links as well.

The fact that we use this exhibit for these multiple but specific purposes limits the versatility of people's actions with it, but not as severely as one might imagine... True, we have not made provision for the visitor to vary the swing periods of the table or the pendulum... On the other hand, we have not designed out all possibility of variation. Although the pendulum swings most readily at right angles to the table, it can also swing parallel to the motion of the table with a very different and not commensurate period. (*ibid.*, p. 15.)

There are two dimensions of versatility. An exhibit can fit into several planned curricula, and it can also lend itself to serendipity, to play, to exploration outside the curricula. Both are important criteria at the Exploratorium. Oppenheimer discusses the technical details that make a particular exhibit versatile in these ways, but there is a more general point that may be more relevant to other forms of education. A precondition to the versatility of *any* exhibit is that the entire museum is pedagogically permissive. You cannot "flunk Exploratorium." Perhaps this is a necessary difference between a museum and a school; perhaps the two kinds of institutions must promote different kinds of learning. But perhaps there could be ways to bring the versatility criterion of the Exploratorium into the schools. Certainly the computer is a very versatile instrument. It can be used for games, for calculation, for revising English papers. (Some of my high school students were once taking a chemistry course in which the teacher told them they could bring one sheet of notes into the final. Our computer had a printer with variable-size letters. These students collaborated to design a tightly packed sheet of notes, printed in the smallest possible size, which they shared with their classmates.) But the versatility of the computer can be squandered if teachers choose to surround its use too tightly with curriculum.

The remarkable feature of the process of individual discovery, whether of detail or of generality, is that the first taste of success can be addicting. For some obscure reason we, as teachers, are committed to turning on addicts. But potential addicts are not programmable;

one never knows who they are or when they are vulnerable. We argue among ourselves: if we do not tell people what they are supposed to find, many will leave with a sense of frustration, but a few will have become addicted to finding more than anybody knew was there. How many frustrated people is one addict worth? Since there is no going back if one gives away too much, we tend to lean toward the more radical answer to this arguable question. And we do have a large number of addicts who come back for more. (*ibid.*, p. 17.)

This is, in an unusual guise, the question about the education of the elite versus the education of everyone. Like Buber, Oppenheimer is telling us to aim high. Buber's reason is that the kind of education that benefits the gifted also benefits everyone. Oppenheimer adds another reason: in a new situation, we cannot always know in advance which student will turn out to be gifted.

The most important aspect of the ambience of the Exploratorium may stem from the fact that visitors are never subjected to judgmental discomfiture. They do not feel compelled to decide whether they are supposed to learn something from an exhibit or merely enjoy themselves. If they stand before an exhibit and say, "Gosh, my eight-year-old child could do that," this remark is made approvingly. (*ibid.*, p. 17.)

Several schools in this study have a computer club, or used to have one, but have not put energy into keeping it active. The computer as the focus of an extracurricular activity could bring this spirit of permission to enjoy oneself into the school. The learning that would ensue might, in the end, be greater than that which comes from a formal curriculum.

Educational Goals: Seymour Papert

Seymour Papert studied and worked in philosophy, mathematics, and psychology before he became involved with computers, so he provides both the wide-angle view of the philosopher and a specific interest in computer-based education. His educational theory and the programming language, Logo, that embodies it have become widely used in elementary schools. My purpose here will be to explore what it would mean to follow this theory at the high school level.

Papert contrasts two kinds of learning: the kind that ordinarily takes place in schools, and what he calls Piagetian learning, the kind a young child does without formal instruction. His long-term goal is to replace school learning entirely with Piagetian learning. “This obviously implies that schools as we know them today will have no place in the future. But it is an open question whether they will adapt by transforming themselves into something new or wither away and be replaced.” (Papert, 1980, p. 9.)

Papert begins his book *Mindstorms* with an anecdote about how he learned to love automobile gears as a child, and how gears served as a recurring metaphor for such things as algebraic equations later in his life. The availability of this already-familiar metaphor helped him learn new ideas. Papert points out that this is what Jean Piaget calls “assimilation.” He wants to provide all children with similarly powerful metaphors.

A modern-day Montessori might propose, if convinced by my story, to create a gear set for children. Thus every child might have the experience I had. But to hope for this would be to miss the essence of the story. *I fell in love with the gears*. This is something that cannot be reduced to purely “cognitive” terms. Something very personal happened, and one cannot assume that it would be repeated for other children in exactly the same form.

My thesis could be summarized as: What the gears cannot do the computer might. The computer is the Proteus of machines. Its essence is its universality, its power to simulate. Because it can take on a thousand forms and can serve a thousand functions, it can appeal to a thousand tastes. This book is the result of my own attempts over the past decade to turn computers into instruments flexible enough so that many children can each create for themselves something like what the gears were for me. (*ibid.*, p. viii.)

This passage raises two important points. One is that falling in love with an idea is central to Papert's goals. Later we shall see that he uses the computer as a metaphor for human thinking in problem solving, an approach that dismays many people. But we must remember that Papert's idea of the human mind also has this affective side; people are not entirely like computers. The second point is that not everyone is expected to fall in love with computers in the same way. It is the child herself who must *invent* a compelling use of the computer. Some elementary schools allow this process of invention by providing access to computers and by *not* providing a curriculum. But there is now a large number of Logo-based curricula full of sequences of activities for the class to do in unison. In that approach, Logo becomes one more subject for school learning, rather than a way to bring Piagetian learning into the school.

It is relatively easy for an elementary school to allow a child to use a computer without imposing a particular structure on the activity. Elementary schools already have the idea of an "enrichment activity" that some children can be doing in the back of the room while others are doing something else in front. These enrichment activities need not be graded. (Giving grades in computer use is a problem for two reasons; first, the spectre of a bad grade may make a child afraid to experiment, and second, in order to have a basis for assigning a grade, the teacher often imposes uniform activities on the class.) Since the elementary school student is in a single classroom most of the day, the scheduling of an enrichment activity is easy; idle moments can be grasped as opportunities.

The situation is different in the high schools. The student's day is divided into periods; during each period, the student is supposed to be in a particular place for a particular purpose. In order for a purpose to count, it must have credits and a grade associated with it. In a tightly scheduled school, a student cannot use a computer without being enrolled in Computer class. That class has a set curriculum, just like every other class in the school.

The five schools in this study all allow drop-in use of the computer facilities before school, after school, or during lunch. But drop-in use that is not directly related to a course assignment is rare in all of them. One reason is that students are kept very busy with assigned, graded work; in the credit economy of the school, they cannot afford to spend time for which they get no credit return. Another reason is probably psychological:

although drop-in use is allowed, it does not seem to be very important to the teachers. It is not considered serious. In the elementary school, the child's use of the computer is not supervised by the teacher, but it is endorsed by the teacher. A third reason is that high school computers do not, as a rule, provide a rich collection of objects for manipulation—the equivalent of Papert's gears. The elementary school computer with Logo provides a "turtle graphics" display; it may provide multiple animated graphic objects; it may provide a physical robot turtle that can move around the floor; it may provide the ability to play music. The high school computer is generally dedicated to more "serious" work, and lacks these facilities.

Short of abolishing courses, class periods, and grades altogether, there are still things a high school could do to get closer to Papert's ideal. It could offer a course with credit, but with no grade, like physical education courses at many schools. This course would provide an official time during which its students would have access both to a computer and to a teacher. There would be no curriculum and no homework, tests, or grades. The teacher would be available for help, if a student asks for it, but would also look over shoulders and offer suggestions when that would be helpful in the particular situation of some student. (I have taught a course much like this in a high school.) This is not a perfect solution; not every student would thrive intellectually in the environment I am suggesting. But it would be one possibility to offer among others.

But "teaching without curriculum" does not mean spontaneous, free-form classrooms or simply "leaving the child alone." It means supporting children as they build their own intellectual structures with materials drawn from the surrounding culture. In this model, educational intervention means changing the culture, planting new constructive elements in it and eliminating noxious ones. This is a more ambitious undertaking than introducing a curriculum change, but one which is feasible under conditions now emerging. (*ibid.*, p. 31.)

Moving away from curriculum is a social, organizational, and political problem. Providing intellectual building blocks is a technical, creative problem. The most famous example of such a building block is the Logo turtle. It is a good example because it allows children access to ideas that were formerly abstract and formal, but that become concrete in the presence of the turtle: angle, shape, scaling, and so on. Papert refers to Piaget's division of

children's thinking into a "concrete" stage and a "formal" stage. One of Papert's goals is to shift the boundary between these stages by providing concrete metaphors for ideas that now lack such metaphors.

To an extent, this issue of providing concrete metaphors is less urgent for high school students than for the younger children Papert has worked with primarily. Most high school students are already at the formal stage, as tested by Piaget, and are less dependent on concrete models than younger children. In particular, Papert emphasizes the fact that a child can "play turtle," using her own body to follow the motion of the computer turtle, and thereby make the turtle's motion much more concrete. I suspect that high school students are much less dependent on being able to use their bodies to visualize a problem. But there are types of problems for which even many adults never develop an intuition. Studies of college physics students, for example, show that many cannot visualize a problem that does not fit in with the specific types of problems they have been taught, even though the same basic laws of physics apply. One computational tool for studying such problems is the "dynaturtle," a Logo turtle that obeys Newtonian laws of motion. The dynaturtle is not a curriculum; it is a programmable object. High school or college students can explore it just as younger students can explore the traditional turtle. More such tools are needed in other areas. At present the dynaturtle is virtually the only example available for older learners.

Two fundamental ideas run through this book. The first is that it is possible to design computers so that learning to communicate with them can be a natural process, more like learning French by living in France than like trying to learn it through the unnatural process of American foreign-language instruction in classrooms. Second, learning to communicate with a computer may change the way other learning takes place. The computer can be a mathematics-speaking and an alphabetic-speaking entity. We are learning how to make computers with which children love to communicate. When this communication occurs, children learn mathematics as a living language. Moreover, mathematical communication and alphabetic communication are thereby both transformed from the alien and difficult things they are for most children into natural and therefore easy ones. The idea of "talking mathematics" to a computer can be generalized to a view of learning mathematics in "Mathland"; that is to say, in a context which is to learning mathematics what living

in France is to learning French. (*ibid.*, p. 6.)

Most high school computer curricula are designed around either or both of two educational purposes: to study computer programming as an end in itself (or in preparation for computer science in college), or to use the computer as a practical tool, as in word processing. Papert wants to use the computer as an *intellectual* tool, to help teach and learn mathematics. But he does not mean the traditional school mathematics curriculum; he means thinking like a mathematician, doing original mathematics. This is one way of interpreting the popular but sometimes vaguely understood goal of teaching “problem solving.”

This goal is perhaps more familiar in the high schools than in the elementary schools. An experience like the one Papert wants has been available for a long time to certain high school students, the ones who join the math club and explore “recreational mathematics.” But until now, the task of learning the language in which recreational mathematics is done has been too hard and too unrewarding for most students. The computer provides a new language in which mathematical exploration is possible.

The critic is afraid that children will adopt the computer as model and eventually come to “think mechanically” themselves. Following the opposite tack, I have invented ways to take educational advantage of the opportunities to master the art of *deliberately* thinking like a computer, according, for example, to the stereotype of a computer program that proceeds in a step-by-step, literal, mechanical fashion. There are situations where this style of thinking is appropriate and useful. Some children’s difficulties in learning formal subjects such as grammar or mathematics derive from their inability to see the point of such a style.

A second educational advantage is indirect but ultimately more important. By deliberately learning to imitate mechanical thinking, the learner becomes able to articulate what mechanical thinking is and what it is not. The exercise can lead to greater confidence about the ability to choose a cognitive style that suits the problem... And giving children the opportunity to choose one style or another provides an opportunity to develop the skill necessary to choose between styles. Thus instead of inducing mechanical thinking, contact with computers could turn out to be the best conceivable antidote to it. And for me what is most important in this is that through these experiences these children would be serving their apprenticeships as epistemologists, that is to say learning to think articulately about thinking. (*ibid.*, p. 27.)

Papert uses the metaphor of “teaching the computer” to describe programming. The power of this metaphor is that it encourages the child to think about teaching another person, or teaching oneself, as something that might be as easy to understand as teaching the computer is. The classic example of a computer-based epistemology for children is the idea of debugging.

[M]any children are held back in their learning because they have a model of learning in which you have either “got it” or “got it wrong.” But when you learn to program a computer you almost never get it right the first time. Learning to be a master programmer is learning to become highly skilled at isolating and correcting “bugs,” the parts that keep the program from working. The question to ask about the program is not whether it is right or wrong, but whether it is fixable. (*ibid.*, p. 23.)

Most high school programming teachers do understand the idea of debugging, and teach it to their students. The trouble is that debugging applies only to the writing of a computer program. When it comes time for the final exam, the student is back in the world of getting it or getting it wrong. To Papert this is a betrayal; the central purpose of learning to program a computer is to change the way we think about our thinking processes in *all* situations, not just in programming. Just as it makes sense to debug a computer program, it makes sense to debug a child’s ideas in every area, rather than to mark them wrong with a red pen.

Perhaps this example shows the reason for Papert’s claim that schooling itself cannot survive the computer revolution. Schools can absorb the computer by turning it into another piece of traditional curriculum. But schools as we know them cannot absorb the computer *culture* that Papert wants to create. If mistakes are not shameful, then any system of exams and grades that penalizes mistakes *is* shameful. Learning without curriculum, ultimately, means learning without classes, without having to be in a particular place at a particular time. Teaching is not obsolete; adults teach infants how to talk, even though not by explicit lessons. But it will have to be a very different kind of teaching.

Conclusions

Here is a summary of the five schools visited in this study.

School	Description	Computers	Stated goals	Some/all
A	literacy, BASIC, Pascal, Assembler	Apple, IBM, Commodore	computer literacy, problem solving	sequence, all $\rightarrow$ some
B	electronic bulletin board, BASIC	Apple	computer literacy, problem solving	course for all plus indep. study
C	Pascal, BASIC, hard disk network	Apple, HP	computer literacy, problem solving	some
D	data processing, BASIC	Commodore	computer literacy, problem solving, vocational ed	all
E	literacy, BASIC, Pascal, Adv. Place., word processing	Commodore, IBM	computer literacy, problem solving, vocational ed	multiple tracks

The columns headed “description” and “computers” are simply for reference, to remind the reader which school is which. The last two columns are the ones of interest in drawing conclusions from the study.

Stated and Implicit Goals

The reader has probably noticed a certain pattern in the goals stated by teachers at each school. Some schools have vocational education programs, while others do not; this difference aside, every school in the study advocates essentially the same goals. (In the case of school C, the two goals were stated by two different people.) Yet the schools are very different from one another in their practice.

Some of the differences are due to factors other than goals. Sometimes the available equipment drives the curriculum instead of the other way around, as in the reappearance of the old Computer Math class at school C because of the donation of computers that only speak BASIC. Sometimes the demography of the school is a factor: probably one reason that programming is less central to school B’s curriculum than to schools A and C is that a smaller fraction of school B’s students go on to college. Sometimes the operation of the computer center is controlled by policy decisions made for the entire school; for example, the loose drop-in atmosphere at school C is possible only because their students have free periods during the academic day. At school A, by contrast, students are always supposed

to be somewhere. The discipline policy in school E's programming classes is driven by a central administrative policy on discipline. Sometimes money is the determining factor, as in the shortage of textbooks at school D.

Some of the practical differences, however, are due to differences in implicit goals. For example, the strong concern for sex equity of the teacher in charge of school B's computer facility has a striking effect on the population of the room. School E's desire to offer an Advanced Placement course affects both what is taught in the lower-level courses and to whom it is taught. Sometimes the atmosphere of the computer facility is strongly affected by the *private* goals of a teacher, such as the desire of school B's Science department head to operate a bulletin board and the desire of school C's Mathematics head to learn Pascal.

Vocational Education

The goals that are actually expressed by the teachers interviewed merit some analysis. I do not have much to say about vocational education, except to make clear what it is: a course of study directed at certain students who have specific career plans in a particular job area, such as word processing. It is not vocational education to teach word processing to every student, either for his immediate use in school or as a disguised form of computer literacy. It is not vocational education to teach an assortment of skills, facts, and ideas not tied to a particular job. True vocational education, such as the word processing course at school E, raises no new philosophical issues when there is a computer involved; there are, of course, issues raised by any form of vocational education, such as the political implications of tracking.

Computer Literacy

Teachers at every school mentioned the idea of computer literacy. I have argued elsewhere that this is an illusory goal. (See Appendix A.) For one thing, it is too vague to serve as a guide to practice. School A's literacy course is largely composed of isolated and sometimes incorrect facts with little relationship to actual computer use. School E's literacy course is almost entirely a course in BASIC programming.

More important, though, is the fundamental error contained in the very phrase “computer literacy.” This is not just an arbitrary name; it is a metaphor. It leads the people who use it to imagine that the content of the computer culture is similar to the content of the written culture, that a person who does not use computers is cut off from the accumulated wisdom and knowledge of humanity in the same way that a person who cannot read is cut off. This is simply not so. Nor is it the case that someone who does not use computers is unemployable.

Problem Solving

The other universal goal is problem solving. I do not consider this goal too vague, but, as Frank Oppenheimer pointed out, it is not easy to know how to proceed toward it. My conclusion from the school visits in this study is that teachers do not really take the goal seriously. For example, the goal is generally expressed not just as teaching students how to solve problems in the context of programming a computer, but also to encourage students to transfer the same ideas about problem solving to other parts of their lives. Taking transfer seriously would mean not teaching programming in isolation, but rather to design a problem-solving course in which a unit on computer programming would be combined with a “critical thinking” curriculum like the ones that have been designed for mathematics classes (Harnadek, 1976) and with explicit application of problem-solving ideas in other domains (Polya, 1957). Seymour Papert has studied attempts at transfer such as teaching children to juggle using the ideas from Logo programming. But I know of no such work in the high schools.

Another aspect of the failure to take problem solving seriously is the use of BASIC as an instructional programming language. The schools that teach both BASIC and Pascal show varying degrees of embarrassment about their continued use of BASIC. They attempt to teach “structured BASIC,” but the lack of locality in the design of the language turns the structured approach into a fetish with no real power. You can write a physically separated subroutine in BASIC, but not a truly modular one. Various people have proposed various improved versions of BASIC. As Seymour Papert says, this “misses the point of

my complaint. Of course one could turn BASIC into Logo or Smalltalk or anything else and still call it 'BASIC.' My complaint is that what is being foisted on the education world has not been so 'fixed.' Moreover, doing so would be a little like 'remodelling' a wooden house to become a skyscraper." (Papert, 1980, p. 220.)

One reason that teachers continue to try to use BASIC as a vehicle for problem solving is that what they see as the alternative, Pascal, has its own problems. As a compiled language, it is hard to use. More importantly, it promotes one particular style of problem solving: top-down structure. In this programming discipline, the programmer is expected to have the entire solution clearly in mind before actually carrying out the steps of implementing it. This approach is analogous to the presentation of a mathematical proof in a formal paper after it has been invented. But it is not very much like the exploration mathematicians do while they are trying to discover the proof. I have argued elsewhere (Harvey, 1982; Harvey, 1984) that of the languages currently available for microcomputers, Logo is the best choice for problem solving. Logo is not the only possible choice, but a teacher who is serious about problem solving as a goal cannot choose BASIC.

Serving Some, Serving All

I have not yet discussed the final column in the chart above. In looking for a way to make a coherent whole of my impressions from the five school visits, the thing that most struck me as a central way to understand each school's computer facility is to look at where the faculty stands in relation to the tension between meeting the computing needs of every student and meeting the needs of the minority of students with a special interest in computer programming. The five schools are quite different from one another in this respect, and the differences are closely tied to all the other differences in style among the schools.

School A operates in terms of a continuum of levels of interest and skill in computer programming. It offers a sequence of courses. In principle, every student starts with Computer Discovery, although in fact some have an equivalent course in junior high school. The alumni of this course progress through BASIC 1, BASIC 2, Pascal, and Assembler;

some drop out at each step. Like Oppenheimer, the faculty attempts to teach each course as though the students were going to progress to the next. It is up to the individual student to decide how far she wants to travel down this road.

School B has a two-level system. The primary focus is on a course for everyone. For the exceptional students who want further instruction, independent study is offered. Everyone starts at the beginning; relatively few continue after the first year.

School C, despite the wishes of the facility manager, has taken a curricular position clearly in favor of serving the interested few. Only programming courses are offered; the emphasis is on Pascal; few sections are taught. Fewer than 10% of the students ever take a computer course. The Pascal courses use a college-level textbook.

School D, somewhat despite the wishes of its teaching staff, has taken the opposite position: its courses are designed to serve the majority of students without a strong interest in programming. The emphasis is on vocations in the computer field.

School E chooses to serve both the many and the few, but it carves up its students into three tracks: vocational, literacy, computer science. The tracks do not form a sequence in the minds of the faculty. There is little overlap even in the teaching staff for the three tracks.

Which of these approaches is best? This is not an empirical question; I cannot settle it by pointing to data from the study. It is partly a matter of values, and partly something that will be different in different schools because of the nature of the student body in each case. Instead of picking a favorite, I shall pose two questions that any of these schools must ask: What *are* the computing needs of every student? And what are the needs of the those students who exhibit, or aspire to, computer programming expertise?

What Does Everyone Need?

I have tried to show, in the second part of this study, that what you think everyone needs in computer education depends heavily on what you think everyone needs in education generally. In the chapters about individual educational writers I have suggested some possible implications of each one's ideas for educational computing. Here is a brief summary

of the goals I found in each writer's work:

John Dewey

learning through personal experience
 mature guidance faithful to experience
 continuity of experience
 collateral learning—learn the desire to learn
 interaction between objective and internal factors
 activities that promote social organization
 democracy as educative society
 reflection on experience

Martin Buber

freedom for the originator instinct
 train instinct through sharing in an undertaking
 entering into mutuality
 freedom as a bridge to communion, not as an end
 genuine education is character education
 relationship of trust between student and teacher
 aim at the great character

Frank Oppenheimer

teach appreciation of science
 understanding is fun
 prepare for further learning
 the power actually to do something
 the value of play
 addiction to discovery

Seymour Papert

Piagetian (curriculum-free) learning
 the learner must love the object of learning
 the learner invents the task
 teachers are providers of building blocks
 make abstract ideas concrete through computer metaphors
 learn math by living in Mathland
doing math, not learning *about* math
 computer as metaphor for human problem solving
 teach children epistemology

It is not my purpose to single out one of these goals as the best. I find all of them appealing. Each teacher will find some of these goals more in tune with his own character than others. For example, not every teacher would want to enter into the intensity of personal relationships with students that Buber talks about, and I think not every teacher need do so. But every child should have one such teacher.

There is a technical as well as a philosophical side to meeting the computing needs of every child. Here, too, schools have not been serious. How many computers, how many printers, would be needed for a school to allow every child to write every English paper with a word processor? To teach word processing in a “literacy” course and then not provide this capability is to make word processing one more abstract, useless piece of school knowledge instead of a practical skill.

What do experts need?

Experts need apprenticeship. That is, they need real work, real tools, and a master who knows more than they do about programming. All of these things may be beyond the economic capabilities of a high school; perhaps they should be provided in regional centers.

One thing that is *not* a special need of experts is training in problem solving skills. Instead, it is the weaker students, the math-haters, who are most in need of this part of computer education. But those students are not the ones who show up in Pascal class. There is something upside-down about concentrating the problem solving curriculum most heavily on the people who need it least. This is another reason why the BASIC/Pascal dichotomy is not a good idea.

Experts need training in the ethics of computer use. When schools address this issue at all, they generally do so in the “literacy” course. Yet it is the expert student who is most likely to harm himself or others through computer abuse. Buber tells us, rightly, that a lecture on ethics in the Pascal class will not serve. Advanced students must be part of a community of work in which ethical issues will arise naturally. (See Appendix B.)

Above all, experts need passion and excitement. Computer programming is an intellectual pursuit that can offer some young people the same feeling of pride in accomplishment

that others get in varsity sports. This opportunity should not be reduced to a set of exercises in a book, in preparation for an exam. Extracurricular programming may be more vital than official coursework.

Design of a Computer Center

To conclude, I shall present a possible design for a computer center that takes account of some of the goals from the philosophical literature reviewed above. The intent is not to suggest that there is one single best design for such a computer center. Instead, the purpose of this concluding section is to indicate how philosophical goals that are expressed in general terms can be interpreted in terms of the specific issues that arise in computer education. This design is personal and idiosyncratic. I take some ideas from the schools visited in this study, but I do not mean to imply that the study “proves” the soundness of the design.

I begin with Dewey’s concern for activities that promote social organization and Buber’s idea of sharing in an undertaking. Computer programming and computer use can be solitary activities that promote isolation rather than group cohesion. To avoid this potential problem, the computer facility should be one that encourages communication and sharing. I shall argue that this criterion supports the use of a timeshared mainframe or large minicomputer, rather than individual personal computers, even on a network.

There are several educational advantages to a timesharing system. One is that the sharing of file storage makes possible a community of programming effort. Not only can students work together as a team, but the program libraries of past students are available to future students. For example, the first few students who want to write video game programs must design procedures for positioning text and graphics at variable points on a display screen. Once these procedures have been written, the second generation of students can use them as tools to build different game programs.

Related to the question of a programming community within the school is the way the school’s computer center fits into a larger programming community. The social style most common in microcomputer programming is that of secrecy and commercialism. Programs

are distributed on copy-protected diskettes. Only executable binary versions are available, not the source programs that would allow students to examine and modify the mechanism of a program. In the large computer world, at least for those computers that are heavily used in university research environments, the atmosphere is very different. Researchers donate programs to software-sharing groups for general distribution, with source files included. The goal is not to make money, but to advance the state of the art. High school students can both benefit from and contribute to this software sharing. It is not only the free software that is valuable, but the social experience of receiving such a gift.

Electronic mail is another way in which a shared-file computer system can help build a human community. School B's bulletin board is a good attempt in this direction, but its hardware limitations restrict its use to one person at a time. Adolescents with access to electronic mail quickly become heavy users. They often use the computer to send messages even to someone at another computer terminal in the same room, if the message is about something emotionally charged, too embarrassing to discuss face-to-face. This is a two-edged sword: such computer-mediated discussions can allow a great degree of openness among students, thereby improving their relationships even off the computer, but such discussions can also come to replace more intimate forms of social contact. To guard against this danger, I would organize computer center social events, from Chinese food expeditions to roller-skating parties.

In principle, a network of personal computers with a central file server could provide file sharing and electronic mail. In practice, however, the existing networks for inexpensive computers do not have the necessary flexibility in the allocation of disk space. Instead, they divide the disk into a small number of relatively large partitions, each assigned to a particular user. Using files in several partitions at once is not easy, and the number of partitions limits the number of user accounts. Also, networked microcomputers still do not display the software-sharing community spirit of the university research systems.

I turn now to Oppenheimer's point about the value of play and Papert's idea of curriculum-free learning. These ideas speak to the question I raised in discussing the schools visited in this study about the tension between the needs of all students and the needs of those who are really interested in computers. I think that the needs of all students

are not well served by joining the “computer literacy” bandwagon, which tells students who have no interest in computers that they had better study computers anyway.

I think that the real needs of all students take two forms. First, the computer must be made available as a tool for immediate practical purposes, most notably as a word processor for students’ writing assignments. To use this tool, a student does not need a semester course. University computer centers typically offer one-week or two-week short courses, not for credit, to introduce students to word processing tools. In a high school, one way to achieve a similar effect would be to include a *brief* word processing unit in the curriculum for freshman English. It would then be up to the individual student to decide whether or not to pursue further instruction.

The second need of all students is to have the opportunity to discover a potential for enjoying computers. School C has a one-time tour of the computer facility for all freshmen. Another approach would be periodic Open House days, when regular class use of the facility might be suspended to make room for visitors, who could run demonstration programs written by students, including games.

But the bulk of my effort would be directed toward the students who express a real interest in computer programming. The centerpiece of the course schedule would be something called Computer Programming Lab, a vehicle to provide students with reserved computer time and course credit for projects they would initiate themselves. This course should be ungraded, so that there is no penalty for an experiment that leads nowhere. The role of the teacher is to offer technical advice within the context of the student’s own project direction. A student who is undertaking a substantial project in a modular programming language can be shown the value of structured programming ideas through the authority of the task itself, as Dewey suggests, rather than through rules imposed arbitrarily by means of the teacher’s power to assign grades.

One direct approach to promoting the educational value of play is through the use of computer games. Students should be encouraged to *write* game programs and to play games written by their peers. This encouragement will have several positive consequences. One is that students will have users for the programs they write; they will feel that their work benefits the community. Also, the users of the programs will have suggestions for

improvement. A program author can deal with such suggestions in two constructive ways. One way is to modify the program as suggested, an experience that will teach about the idea of human engineering of software; another is to tell the complaining user to fix it. In this latter case, what results is a master-apprentice relationship *between two students*. Another benefit of widespread computer gaming is that students with a latent interest in programming will have the opportunity to see that other students like themselves have written successful and entertaining projects. The use of a timesharing system allows the development of multiplayer games, which promote social interaction and work against the asocial atmosphere of the usual video games.

Buber talks about the human relationship between teacher and student, and the human qualities required for a good teacher. I believe that the teacher is the most important ingredient in a successful computer center. The teacher's humanity may be of greater concern than ever when the object of study is a machine. But in particular, it should be remembered that the teacher-student relationship is actually a triangle, with the common undertaking (running a computer facility) as the third vertex. The teacher should have a personal interest in computer programming, and should be truly expert at some branch of the field. The teacher's interests will affect the directions chosen by many students. My own background is in operating system development, so my students tend to be system programmers. There is nothing uniquely educational about system programming compared to any other kind, but there *is* something educational about having some particular direction that makes one school's computer center different in tone from the rest. Another teacher might be excited by artificial intelligence, for example, or telecommunications, like the one at school B. Whatever the field, I have seen in my own work that a high school teacher of computer programming can successfully follow the model of the university professor who pursues his own research interests, attracting like-minded graduate students as colleagues in the research. Such a collaboration follows Dewey's recipe of "more multiplied and more intimate contacts between the mature and the immature than ever existed in the traditional school, and consequently more, rather than less, guidance by others."

Appendix A

Stop Saying “Computer Literacy”!

[A version of this appendix was published in *Classroom Computer News* (Harvey, 1983).]

As computers become more widely used in schools, educators have invented many different ways to use them. The different uses for computers are based on quite diverse models of what the purposes of computer education should be. One very common approach is based on the notion of “computer literacy”: the idea that there is some basic familiarity with computers which *all* students need in order to compete in the job market, or to be informed citizens. The purpose of this appendix is to question whether that notion is valid.

The Importance of Metaphor

Several educators have questioned the specific details of particular “literacy” plans, while accepting the overall concept. “Computer literacy doesn’t mean X,” they say, “it means Y.” For example, Arthur Luehrmann says, “[C]omputer literacy must also mean the ability to *do* computing, and not merely to recognize, identify, or be aware of alleged facts about computing.” (Luehrmann, 1981) His argument is that a computer curriculum should give students actual programming experience, not just reading about computers in a book. But Luehrmann accepts the concept of computer literacy itself, the idea that there is *something* which every student must learn. He questions only the details of just what this something is.

What I would like to do is call into question the claim that there is anything about computers which belongs in the school experience of *every* student. I agree with Luehrmann that programming can be a valuable experience. But the word “literacy” means more than that; it means that everyone must have the experience in order to be able to function at all in society.

Many people, hearing me make this argument, have complained that it is foolish to argue about a word. “Call it something else, then, and let’s get on with deciding what it should be.” This complaint misses the point, and indeed could be said to prove the point.

People who have become accustomed to the “literacy” idea find it very difficult even to entertain the question of whether there is any universally required computer experience.

The trouble is that the word “literacy” is a magic word, which conjures up a very strong metaphor. Literacy in its original sense, knowing how to read, really *is* universally required in our society. Any educator who suggested eliminating reading from the curriculum would be laughed at, if not tarred and feathered. Merely to say the phrase “computer literacy” definitively answers a question which has not explicitly been asked. It’s like the classic “When did you stop beating your wife,” but harder to recognize because it is so widely used.

One practical result of the literacy metaphor is that many decisions about computer education have been made in a kind of panic. Parents call up the school committee to ask why their children are not being trained for the vital computer job skill. These parents may not know just what that skill is, and neither does the school committee. But they do know that the private school down the road has computers.

What Everyone Needs to Know

It’s quite true that more and more jobs involve the use of computers. That’s different from saying that these jobs involve computer *expertise*. Let’s look at some examples.

A good example of a computer-using job is that of selling hamburgers at McDonald’s. That machine behind the counter which looks like a cash register is actually a computer terminal. When you order a Quarter-pounder with cheese, instead of ringing up whatever the price is, the counter person pushes a button which says “Quarter-pounder with cheese.” The computer displays the price, but it also keeps track of how many of this item are sitting under the heat lamps. If they’re running short, the computer tells the cooks in the back to make more. The same computer can tell the manager to buy more hamburger buns at the right time. It’s a pretty sophisticated system, and helps make McDonald’s a low-overhead operation.

Now, what does the person behind the McDonald’s counter need to know about computers? He only needs to know that when you ask for a Quarter-pounder with cheese, he

should push the button which says “Quarter-pounder with cheese.” That’s it. Nothing about input unit, output unit, processor, and memory; nothing about programming either.

Perhaps a more common example is that of computer word processing. The word processor involves a much more intimate interaction with the computer than selling hamburgers. Still, the manufacturers of word processing systems take pains to *hide* the computer, to make the way you operate the machine as much as possible like operating an ordinary typewriter. What skills does a word processing operator need? For the most part, exactly the same skills a secretary needed before word processing: good spelling and punctuation, touch typing, being able to read the boss’s handwriting. What the computer adds is mostly a matter of pushing the “paragraph” button instead of the carriage return and the tab.

There are more specifically computer-related skills, of course, like threading the fanfold paper into the printer. But that’s the sort of thing which is done differently for each specific model, and which is part of a very specific job training program. It isn’t what you’d teach in a “computer literacy” course for everyone.

Perhaps the point really is that there are too many *different* computer-related skills to teach all of them to every student. Using a spreadsheet program, for example, is very different from using a word processor. Using a microprocessor-controlled automobile engine tester is very different from either. Neither “this is the return key” nor actual programming will help much with any of those job activities.

An Analogy

Virtually everyone in our society deals with automobiles. Each of us would do well to know about what goes on under the hood, and many people do. But many don’t, and they function okay. When the car breaks down, they call the AAA. Some people drive in unusual conditions, far from help, and those people may need to be self-reliant in the field of automobile repair. But it would be silly to require an auto mechanics course for every student. It would be even sillier to require an “automobile literacy” course in which students read about cars without using one. Many students do find driver training worthwhile, but even that isn’t required. Some people don’t drive; some grow up on farms and start driving

tractors long before the official driver training age; some are taught by their parents.

Reasons for Computer Literacy

I think the notion of computer literacy grew up in the first place as an accident of history, because of the great speed with which computers have become important. Curriculum developers are, by and large, 30 or 40 or 50 years old. They grew up in a world without computers. Suddenly, they have had to face up to a shocking change in the very nature of the world around them. In inventing computer literacy courses, they have designed a curriculum *for themselves*, not for the kids who have grown up with Pacman, automated bank tellers, and bar code readers in the supermarket.

Another, more sophisticated reason is a confusion about the role of education in social mobility. Many people think that by learning to program computers one can get a good job. The problem with this idea is that what works for a few people stops working when everyone does it. Many years ago, hardly anyone had a high school diploma. Those who did have one could get the cream of the jobs. A few decades ago, about half the population of the United States graduated from high school. They didn't get the very best jobs—by then, you needed a *college* diploma for that—but they did get decent and fairly secure jobs. Now, almost everyone has a high school diploma, and it's worthless.

Similarly, the day of the matchbook-cover computer school as the route to riches is already over. There are many openings for computer programmers, but the jobs go to people with Computer Science doctorates, preferably with experience. There is still room for the programming entrepreneur, but the people who succeed in that role are not the product of computer literacy courses. They are computer enthusiasts who do it *because it's fun*, not because it's homework.

I believe that the effect of the “computer literacy” movement will be the opposite of what is intended. In the early days, when very few schools offered such courses, their alumni really did have a competitive edge in the job market. But as we approach the day when every school offers “literacy” courses, a day eagerly sought by those who believe in the literacy metaphor, there will be more “literates” than jobs. Those who have taken the

courses won't find easy jobs. But those who *haven't* taken the courses will be completely out of luck. Ironically, the literacy metaphor will become self-fulfilling, not because there is any real need for such training, but because employers will start demanding it as a filter for job applicants, just as there is no *real* basis for requiring a high school diploma for many jobs today.

It used to be that computer companies like IBM ran their own training program for their employees. A friend of my family was a high school dropout who was hired by IBM as a keypunch operator, and through IBM's training courses learned to program. She is now a very successful senior programmer. That sort of opportunity isn't available any more; you need both a diploma and data processing experience to get hired at IBM. The poor and the black, who are disproportionately high school dropouts, will find one more barrier to employment as "computer literacy" joins the list of school subjects which some people pass and some fail.

Another Analogy

For a moment, forget about the idea of survival skills, like reading, and think instead about the high school newspaper. Some kids work on the newspaper and some don't. For those who do, it is an incredibly valuable experience. The value is *not* realized only for those students who go on to become professional journalists. Everyone who participates learns writing skills, critical thinking skills, and the subtler skills of meeting deadlines, getting the job done, depending on themselves instead of on some adult. I would recommend the experience to any young person.

But those who don't join the newspaper staff aren't unemployable. There are other ways to learn the same skills. Also, some people may not learn those skills at all, but may learn different skills, in a different setting, perhaps even out of school.

Now imagine that for some reason there grew up a national movement for "journalism literacy." Need I describe the results? Aside from the economic and political disaster I've already mentioned, the quality of the newspapers themselves would go down. The learning experience would become a much less valuable one, intrinsically, job status aside.

So What Should We Be Doing?

If computer literacy is the wrong idea, what's the right idea?

One right idea, I think, is suggested by the newspaper analogy. The computer can be an appealing medium for learning for some students, without being forced upon every student. Computers are a general-purpose tool; they may appeal to different students in different ways. One student may like graphics and animation. Another may like electronic mail. Another may prefer video games. The architecture and scheduling of a computer facility should encourage all these diverse uses.

Another right idea is to make the computer available to students as a serious tool, in their lives *right now*, not as something they'll need later. Probably the most important example of this approach is word processing. Students have to write many papers, for English teachers, history teachers, and so on. Word processing can make the mechanics of this task much easier, encouraging second drafts. Instead of setting up a required course, try just letting students know that this facility is available to them if they want it. But provide enough printers to handle the load!

Finally, for students who are interested, the computer can be valuable not as narrow job training, but as a medium for developing problem-solving skills. It can also be a medium for developing independence, since not every student need be doing the same programming project at the same time. This educational use of computer *programming* is the central idea behind the Logo approach to computer education (Papert, 1980).

Conclusion

What's in a name? A great deal, when the name brings in a strong hidden assumption about the sociology and economics of the job market. "Computer literacy" implies some skill or knowledge which is necessary for every person to be able to cope with the computer-centered society. This implication leads to certain specific policy decisions about computer education; for example, it leads to spreading out the available computer time (of which there is never enough) among a very large number of students, some of whom aren't interested. The ones who *are* interested then don't get enough time to pursue their interest in any

meaningful way.

The model implied by the literacy metaphor is false. There are *no* universally valuable computer skills, or universally important facts about computers. Changing the definition of “computer literacy” to contain more programming experience, or more study of social implications, or more study of binary numbers, is missing the point.

The bad drives out the good. As more and more schools adopt the “literacy” metaphor, other schools are under terrible pressure to go along. The day will come when the metaphor becomes self-fulfilling. To prevent that, we must decide explicitly to reject this model for computer education.

Appendix B

Computer Hacking and Ethics

[This appendix is excerpted from a paper presented to a panel on “The Ethics and Morals of Hacking” (Harvey, 1985).]

[Neal Patrick] said he and his friends, who named themselves the “414s” after the Milwaukee area code, did not intend to do any damage and did not realize they were doing anything unethical or illegal. In fact, when asked [at a Congressional subcommittee hearing] at what point he questioned the ethics of his actions, he answered, “Once the FBI knocked on the door.”

— “‘Common Sense’ Urged on Computer Break-Ins,” 26 Sept 83;
(C) 1983 New York Times News Service

It’s no secret that a mature sense of ethics is something a person develops over time. Parents are supposed to exercise authority over their children because the children are not expected to know how to make certain decisions for themselves. We have a juvenile court system separate from the adult criminal court system because we believe that a young person is not *capable* of crime in the same sense that an adult is capable of it.

Within this century, the obvious idea that the ethical sense of an adolescent isn’t the same as that of an adult has become the focus of scientific research. Psychologists have entered a field once left to philosophers: moral development. The best-known attempt to formalize this development is probably the six-stage theory of Harvard psychologist Lawrence Kohlberg. Here is his description of Stage 3, the Interpersonal Concordance or “Good Boy-Nice Girl” Orientation:

Good behavior is that which pleases or helps others and is approved by them. There is much conformity to stereotypical images of what is majority or “natural” behavior. Behavior is frequently judged by intention—the judgment “he means well” becomes important for the first time. One earns approval by being “nice.” (Kohlberg, 1981, p. 18)

Is Neal Patrick at this third stage of moral development? He seems to judge his own actions in terms of intention. From the perspective of the stage theory, we can see this as an improvement over “Our mistake was to get caught” or “What have those computer

companies done for me,” responses that would be typical of the earlier stages.

It’s not scientifically valid to assign Patrick to a developmental stage on the basis of one quoted sentence. Also, not every researcher accepts Kohlberg’s stages. But it doesn’t really matter whether or not we use the specific details of Kohlberg’s system. The important point is that Patrick is not some new kind of monster spawned by computer technology; he’s a kid with all the strengths and weaknesses we expect from kids in other situations.

Compare a bunch of adolescents breaking into a computer system with another bunch of kids hot-wiring a car for a joyride. The latter would probably argue, with complete sincerity, that they were doing no harm, because the owner of the car recovered his property afterward. They didn’t keep or sell it. It’s a “naughty” prank to borrow someone’s property in that way, but not really serious.

These hypothetical car thieves would be wrong, of course, in making that argument. They might lack the sensitivity needed to give weight to the victim’s feelings of manipulation, of fear, of anger. They may not understand how the experience of such a random attack can leave a person feeling a profound loss of order and safety in the world—the feeling which leads half our population to hail Bernhard Goetz as a hero to be emulated. Some adolescents don’t have the empathy to see beyond the issue of loss of property. Some may show empathy in certain situations but not in others.

The point is that the computer raises no new issue, ethical or pragmatic. The password hacker who says “we aren’t hurting anything by looking around” is exactly analogous to the joyrider saying “we aren’t stealing the car permanently.”

(The two cases need not seem analogous to an adolescent. There may be many computer abusers who would never break into a car for a joyride, but who don’t understand that breaking into a computer account raises the same ethical issues. But the analogy still holds for us as adults.)

The professional car thief and the teenaged joyrider are both social problems, but they’re *different* problems. To confuse the two—to treat the teenager like a career criminal—would be a disastrously self-fulfilling prophecy.

In the context of computer systems, there is a similar dichotomy. There are some career criminals who steal by electronic means. This small group poses a large problem

for society, but it's not a new one. Thieves are thieves. Just as banks use special armored cars, they must also develop special armored computer systems. But the rest of us don't use armored cars for routine transportation, and we don't need armored computer systems for routine communication either.

Two Models for Moral Direction

What to do about it? Saying that the problems of computer ethics are like other ethical problems doesn't solve them. Many approaches are possible. We are starting to hear among computer experts the same debates we've heard for centuries among criminologists: prevention, deterrence, retribution, cure?

Among all the possible approaches, it may be instructive to consider two strongly opposed ones: first, control of the technology, and second, moral training. As examples of these approaches, compare the registration of automobiles with instruction in karate.

Automobile registration is certainly a good idea in helping the police control professional crime. As thieves have learned to steal cars for their parts, rather than to sell whole, the technology of registration has had to grow more sophisticated: we now see serial numbers on each major component, not just on the door frame. But registration doesn't help against joyriders.

Other technological security measures can help. Steering column locks have made joyriding harder, but not impossible. Many adolescents are expert locksmiths, not because they're dishonest but because locks and keys pose a technical challenge much like that of passwords in a computer system. Also, increased security has made the consequences of juvenile car theft more serious, because the easiest way to defeat a steering column lock is to destroy it by brute force.

The example of karate instruction shows a very different approach to the problem of adolescent moral limitations. Instead of using technology to limit the power of young people, this second approach deliberately empowers them. Skill in karate is a deadly weapon; to give that weapon to a young person is an affirmation of trust rather than suspicion.

Why do karate classes for kids work? Why don't they lead to an epidemic of juvenile

murders? This could be an interesting question for psychological research. This paper can't present a definitive answer. But we can suggest some possibilities and use them to draw analogies for computer education.

One probable reason is that every person responds to his or her situation. If I know you're trusting me with something important, I'll try to live up to your trust. If I sense that you consider me untrustworthy, I may decide that I might as well live up to your low expectations.

Another vital reason, though, is that the technical instruction in karate techniques is part of a larger initiation into a certain culture and its rules. Karate schools don't begin by telling novices, "Here's how to kill someone." They begin with exercises to develop discipline and with a solemn oath not to abuse the art. But beginning students know that the exercises aren't just busywork; there is a true empowerment at the end of the road.

Empowerment in Computer Education

How can we *teach* young computer enthusiasts to be responsible members of the electronic community, without defining them as criminals? The analogy of karate instruction suggests that the answer is to combine ethical training with real empowerment. To turn this broad slogan into a practical program requires several changes in our approach to educational computing and to computing in general.

Growth, like any ongoing function, requires adequate objects in the environment to meet the needs and capacities of the growing child, boy, youth, and young man, until he can better choose and make his own environment. It is not a "psychological" question of poor influences and bad attitudes, but an objective question of real opportunities for worthwhile experience.... Thwarted, or starved, in the important objects proper to young capacities, the boys and young men naturally find or invent deviant objects for themselves; this is the beautiful shaping power of our human nature. Their choices and inventions are rarely charming, usually stupid, and often disastrous; we cannot expect average kids to deviate with genius. (Goodman, 1960, pp. 12–13)

Paul Goodman was discussing traditional juvenile delinquents, not password hackers. But the problem is fundamentally the same. How can we provide a worthwhile culture for young

computer enthusiasts to grow into?

1. Serious adult models. In karate instruction, discipline is not only for novices. The adult instructors follow the same discipline themselves. The ethical principles taught to beginners are taken seriously in the adult community. As a result, young students don't see the discipline of karate as an arbitrary imposition on them; they see it as part of what it means to be a full member of the community.

In the computer culture, adults rarely take seriously the idea of belonging to a community. The only ideal is the self-serving entrepreneur. Our heroes are the ones who become millionaires by doing a slick marketing job on yet another spreadsheet program. The conversation over the dinner table is about tax loopholes and padding expense accounts. In this context, why should any young person listen to our moral lecturing?

2. Access to real power. Another important part of the karate analogy is that there are not two kinds of karate, one for adults and one for kids. What beginners learn may be elementary, but it's a start down the same road traveled by experts. The community into which young karate students are welcomed is the real, adult community. That's not how things work with computers. How many adult computer scientists put up with CP/M, BASIC, and floppy disks? The technology available to most young people is not a simpler version of what experts use; it's a completely separate, more arcane, fundamentally less powerful medium. That medium—the programming languages, the file storage, the editing tools, and so on—is simply inadequate to challenging intellectual work.

The community of computer professionals has come to take for granted easy access to electronic communication with colleagues anywhere in the world. Those of us lucky enough to be on the Arpanet have instantaneous communication supported by taxpayers. Even the less fortunate who communicate over dialup networks like uucp, though, have the cost of their mail supported by computing facilities other than their own; the general agreement among even competing private businesses to forward one another's mail is a remarkable example of disinterested cooperation. Some of this mail traffic is serious business. But some of it is also "junk mail" like sf-lovers (for science fiction enthusiasts) and human-nets. Is it surprising that young computer enthusiasts want a slice of the pie too?

Adolescents are excluded not only from access to equipment but also from access

to ideas. The password hackers' preoccupation with magic words and magic numbers is harmful to *themselves* as well as to the rest of us; it's an intellectual dead end that gives them no real insight into computer science. They learn a bag of isolated tricks rather than powerful ideas that extend to solving other kinds of problems. Instead of just telling them what's forbidden, we would do better to show them the path to our own understanding of algorithms, formal theory of computation, and so on. We all know you can't program well in BASIC; why do we allow manufacturers to inflict it on children?

3. Apprenticeship: challenging problems and access to expertise. The karate student is given not only access to a body of knowledge, but also the personal attention of a master in the field. The instructor is responsible for the moral development of his students as well as their technical skill. He steers them in the direction of challenges appropriate to each one's progress, and his own expertise is available to help the learner.

For many years, the MIT Artificial Intelligence Laboratory ran a computer system with no passwords and no file protection at all. (It was pressure from ARPA, not internal needs, that forced them to implement a password scheme.) Even now, the laboratory has a liberal "tourist" policy: anyone can have an account, provided that someone at the laboratory is willing to be his or her mentor. The philosophy behind this policy is that most "malicious" computer abuse is the result of ignorance and misunderstanding, rather than real maliciousness. With a particular person responsible for each new user, tourists learn to share the values of the community. They are taught that the vulnerability of MIT's system is a price researchers pay willingly for the open exchange of information that that vulnerability allows. Treated as legitimate members of the community, even young tourists quickly learn to act responsibly toward the group.

Not every computer facility can be expected to share the vision of MIT-AI. Certainly the computers that control the missiles and the banking transactions should not be so open to visitors. But a typical large company has several computers, not all equally sensitive. Many could allow access to young people in their communities in the evenings, especially if some of their professional staff members are interested in serving as volunteer mentors. It's the mentor/apprentice relationship that makes all the difference. Just giving a kid an account on your machine may be asking for trouble, but making a friend of the kid is a

good investment.

For secondary schools, the issue is partly one of curriculum. Too many teenagers are taught (not only in the schools but also in the magazines) that true computer expertise means knowing what number to POKE into what address in order to change the color of the screen on some brand of microcomputer. Such learning is not intellectually challenging. It does not lead to a feeling of fruitful apprenticeship.

4. A safe arena for moral experimentation. The beginning karate student might be afraid to try his or her skill with a fellow student, lest he or she injure or be injured. But it's safe to fight a match with a black belt instructor. "I won't hurt you," says the instructor, "and I won't let you hurt me."

Young people have a similar need for safety in moral experimentation. One of the reasons for the appeal of role-playing games like Dungeons and Dragons is that a player can say "I'm going to be a thief," or "I'm going to be evil," trying on these roles without actually harming anyone. Similarly, a good school should be a place where students feel safe, a kind of "ethics laboratory."

Neal Patrick's first exposure to an ethical dilemma should not have involved the FBI. He should have confronted the issue of information privacy while using a computer system in his school. He could have learned how his antisocial acts hurt and angered the legitimate users of the system, without risking really serious trouble for himself or for anyone else. For one thing, it's hard for a young person to understand the chain of reasoning from the abstract corporate owner of a computer system to the actual human beings whose lives are affected when that system breaks down. It's easier to understand the issues when the users are one's friends and classmates, and the social effects of malicious password hacking are immediately apparent.

None of this is meant to excuse Patrick or the other 414s. Neither ignorance of the law nor misunderstanding the ethical issues is accepted in our culture as an excuse for lawbreaking. But we are not a court of law meeting to settle Patrick's guilt or innocence. The question for us is how, as a society, we can act to make the next generation of teenagers less likely to paint themselves into this particular corner.

As a practical matter, what's needed to build an ethics laboratory for computing stu-

dents is adequate computing power to support a user community, as opposed to a bunch of isolated, independent microcomputer users. Whether this means timesharing or a network of personal computers with a shared file server is a technical question beyond the scope of this appendix. But sharing is essential. The ethical issues of a living community don't arise in the context of isolated individuals using microcomputers separately with no communication among them.

Appendix C

Course Materials

This appendix contains course descriptions, curriculum units, worksheets, and other handouts from the computer courses at the schools studied. The schools' official goal statements are included as figures in the chapters, as are a few course handouts to which special reference is made in the text.

Appendix C.4. Bulletin board map.

References

- Becker, Henry Jay. "School Uses of Microcomputers," *The Journal of Computers in Mathematics and Science Teaching*,
 Report #1: vol. 3, no. 1, Fall, 1983.
 Report #2: vol. 3, no. 2, Winter, 1983/84.
 Report #3: vol. 3, no. 3, Spring, 1984.
 Report #4: vol. 3, no. 4, Summer, 1984.
 Report #5: vol. 4, no. 1, Fall, 1984.
 Report #6: vol. 4, no. 2, Winter, 1984/85.
- Boyer, Ernest L. *High School: A Report on Secondary Education in America*. New York: Harper & Row, 1983.
- Buber, Martin. *Between Man and Man*. Boston: Beacon, 1955.
- Dewey, John. *Democracy and Education*. New York: Macmillan, 1916.
- Dewey, John. *Experience and Education*. New York: Collier, 1963.
- Goodman, Paul. *Growing Up Absurd*. New York: Random House, 1960.
- Harnadek, Anita. *Critical Thinking*. Pacific Grove, California: Midwest Publications, 1976.
- Harvey, Brian. "Why Logo?" *Byte*, vol. 7, no. 8, August, 1982.
- Harvey, Brian. "Stop Saying 'Computer Literacy'!" *Classroom Computer News*, vol. 3, no. 6, May/June, 1983.
- Harvey, Brian. "Tinker's Misunderstanding of Logo," *Currents*, vol. 2, no. 2, Spring, 1984.
- Harvey, Brian. "Computer Hacking and Ethics." Unpublished presentation to the Association for Computing Machinery panel on "The Ethics and Morals of Hacking," Menlo Park, California, April, 1985.
- Kohlberg, Lawrence. *Essays on Moral Development, volume 1: The Philosophy of Moral Development*. New York: Harper & Row, 1981.

Luehrmann, Arthur. "Computer Literacy—What Should it Be?" *The Mathematics Teacher*, vol. 74, no. 9, Dec. 1981.

Oppenheimer, Frank. *The Exploratorium*, Special Issue, March, 1985.

articles cited:

"Teaching and Learning"

"Adult Play"

"Everyone Is You...Or Me"

Papert, Seymour. *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books, 1980.

Polya, George. *How To Solve It*. Garden City, New York: Doubleday, 1957.

Turkle, Sherry. *The Second Self: Computers and the Human Spirit*. New York: Simon and Schuster, 1984.