SESSION VI

WHAT IF ALL MY STUDENTS HAD PERSONAL COMPUTERS?

A Symposium
THOMAS T. HEWETT, *Presider*

When every student has a computer: A new perspective on courseware and its development

THOMAS T. HEWETT
Drexel University, Philadelphia, Pennsylvania

In this paper, I describe some of the institutional and curricular implications of 100% student access to personal computing. I then explore the new perspective which these conditions create on the availability of courseware, on the process of courseware development, and on future directions in courseware development.

A new challenge and a new opportunity are created by the machine-rich environment that exists when every student has a computer. The challenge is in reexamining the contents of the curriculum and looking for things worth doing with the computer. The opportunity lies in introducing a computer-based component into the entire psychology curriculum, tying together the entire curriculum in new ways which introduce a new kind and level of experience for faculty and students.

The personal computer allows one to create a structured learning environment, not only in the lecture hall and the tutorial laboratory, but also wherever the student keeps the machine. Freedom of access helps to free instructional computing from the capture effect of the computer itself. In a machine-rich environment, it is no longer possible to capitalize on whatever intrinsic motivation effect is created by providing students with access to a scarce resource. In addition, ease of access creates new opportunities for the use of microcomputer application programs (e.g., Hewett, 1985) and for courseware development.

Several ideas presented here were first developed while the author was the recipient of faculty development and courseware development minigrants funded from a grant to Drexel University by the Pew Memorial Trust. In working out some of these ideas, it has been particularly useful to have the opportunity to view and discuss with colleagues at Drexel the wide variety of courseware materials which they have developed, or are currently developing. The final draft of this paper has benefited considerably from the reactions and feedback provided by anonymous reviewers and by colleagues at Drexel who read and commented on earlier drafts.

Reprint requests should be mailed to the author at: Department of Psychology and Sociology, Drexel University, Philadelphia, PA 19104.

A NEW PERSPECTIVE ON COURSEWARE

One cumulative effect of the factors which become salient in the switch from a machine-sparse to a machine-rich environment is to force a new perspective on currently available courseware and upon which new computer-based activities ought to be developed and how they ought to be developed. For example, looking at the availability of courseware, Hewett and Perkey (1984) recently argued that when an educational institution sets a goal of 100% student access to personal computing, there is actually very little in the way of good-quality university-level courseware readily available for instructional use. This claim seems to be true, regardless of the source of the courseware or the personal computer involved.

One of the more obvious factors involved in justifying this claim is cost—both in money and in time—to the student. Does one really want to require students to do computer-based work in an instructional laboratory or tutorial session when the same work can be done more thoroughly and efficiently by a student working on his or her own machine at a more convenient time and place? Does one really want to tack an extra \$50 to \$100 onto the costs for each student for each introductory-level course taken? (In actual practice, many students would not pay this much extra, given the ease of software piracy. However, we do not want to be in the position of encouraging students to steal the product of someone else's creative efforts.)

Another factor, implicit in the arguments made by Hewett and Perkey (1984), is that the amount of time a

student spends in learning while in college is quite large relative to the amount of time that can be filled by the available courseware. For example, suppose that the useful psychology courseware currently available could all be assembled to run on a single computer using a consistent user interface. How many courses from the entire curriculum would be affected? With some judicious picking and choosing, it might be possible to computerize introductory courses in experimental psychology, statistics, and possibly in social and cognitive psychology. In addition, bits and pieces of courseware in several other courses could be put on disk, but that is about all.

A NEW COURSEWARE DEVELOPMENT ENVIRONMENT

What is particularly important about the scarcity of courseware is that in a machine-rich environment, an institution is forced to move away from the situation where there is only grudging recognition, if any, of either the legitimacy or the desirability of creating courseware as part of a faculty member's instructional role. Unlike a machine-sparse environment, where courseware developers have sometimes had to fight for recognition and acknowledgment of their work, courseware developers will find their skills and interests are valued in a machine-rich environment. In addition, developers will find a reasonably large number of colleagues from other disciplines who are willing to explore and share ideas both about techniques and about pedagogical goals and philosophies.

The problem is how to remain a psychologist and a teacher instead of becoming a programmer. Part of the solution to this problem lies in the institutional emergence of a new courseware development environment. The same pressures which force institutional recognition of the value of courseware development force a change in the nature of the courseware development process. These same pressures also force upon courseware developers a more realistic view of the relative importance of the various activities involved in courseware development.

The Two Roles in Courseware Development

In a machine-sparse environment, the beginning courseware author is burdened with two roles: figuring out what the computer ought to be doing and figuring out how to get the computer to do it. This latter problem requires a considerable investment of time in developing the skills needed to implement a good idea. Once the programming skills have been developed, an experienced courseware author may take only one afternoon to develop a program. However, it may take 10 years to really understand how to present the material. There are two key points here. The first is that the courseware developer has to understand the material. The second is that good ideas are in short supply, but programming talent is not.

To redress this imbalance, it is necessary to involve faculty members who have good ideas for courseware development without requiring that they learn the intricacies

of programming. This type of faculty involvement necessitates the creation of a pool of programming talent to provide support in instructional software development. These programmers then work closely with individual faculty members to develop courseware which serves the faculty members' instructional goals and needs. This type of service greatly enlarges the pool of faculty members who can work productively on courseware development without having to risk giving up their disciplinary identification.

Definition of Courseware Development Roles

Given that it is possible to identify two roles in courseware development, how are the roles to be defined and how are the individuals to interact? Anyone setting out to design courseware in a machine-rich environment should read *The Mythical Man-Month* by Frederick Brooks (1975). Brooks, who was manager of IBM's System 360 development project, argued for placing a premium on conceptual integrity of design. This led him to suggest that there be a clear separation of roles in software development. Likening the design and completion of software to the construction of a building, Brooks proposed that a software development project should have both an architect and an implementer. (Somehow, "builder" seems more appropriate than "implementer.") The software architect, like the architect of a building, is the user's agent. "It is [the architect's] job to bring the professional and technical knowledge to bear in the unalloyed interest of the user, as opposed to the interests of the salesman, the fabricator, etc." (p. 45). In this case, however, the architecture Brooks is describing is "the complete and detailed specification of the user interface" (p. 45).

The builder, or programmer, is to have creative and inventive control over the actual implementation of the architect's design. Thus, the builder's job is to bring technical and construction skills to bear in developing a clean, realistic, efficient, smoothly operating product. In part, the discipline which prevents the architect from developing unrealistic and unimplementable specifications is imposed both by the available building materials and by the technical skills and creativity of the builder. In part, the discipline which prevents the builder from adding good ideas or features which do not integrate with the system's basic design concepts is provided by the architect's specifications and requirements. The dynamic tension between architect and builder requires close, detailed communication between them and a mutual respect for the skills, goals, and constraints which govern the options available to each.

The Relationship Between Courseware Architect and Builder

A courseware architect needs to engage in a continuing process of self-education on capabilities, limitations, and possible options for implementation of various generic ideas to be used in courseware development. That is, the architect must understand the nature of the tools and

materials available to the builder. In particular, the architect needs to be continually alert to the possibility that a particular program shell or generic application already available to the builder can be modified to serve an architectural goal of which the builder may be unaware.

The architect should be responsible for development of the initial draft version of project-planning documents to be provided to the builder. These project-planning documents should contain a description of the complete architecture of the courseware, focusing primarily on the nature and sequence of user interaction with the courseware module, and a general statement of the objectives of the programming work to be done. A useful tool in this context is the development and revision of "storyboards." Storyboarding is a technique borrowed from filmmakers. The courseware architect uses storyboards to create a visual representation of the sequence and nature of end-user interactions with the computer by providing "snapshots" of crucial screen displays or choice points in the interaction.

Depending upon the amount of programming time being devoted to the project, the architect should hold regular planning and evaluation sessions. These sessions should be conducted with all individuals working on the project. Part of the focus of these meetings should be upon revisions of the planning document and storyboard which may be necessitated by the new ideas and unanticipated problems which emerge in the working out of the architect's plans.

It is particularly important for the architect to listen to the builder's ideas because they can represent a rich source for improvements to the overall design. However, it is the architect's job to protect the conceptual integrity of the overall design of the courseware module from additions or changes which do not fit, regardless of how good the ideas are (unless, of course, the original design proves to be fundamentally flawed, in which case it should be scrapped and a new one developed).

It is also important for the courseware architect to be prepared to protect the builder. Because programming is often a creative activity involving a great deal of abstract thought, it is easy for the builder to overestimate the ease with which something can be accomplished. This can lead to an investment of considerable time and energy in trying to solve a problem that appears to be quite important to the programmer, but which is relatively minor in the context of the whole project. Sometimes the easiest way to solve the problem is for the architect to redesign around it.

A NEW EMPHASIS IN COURSEWARE DEVELOPMENT

Just as the machine-rich environment forces a new perspective on available courseware and on the courseware development process, it forces a new perspective on instructional computing and the kinds of courseware that

ought to be developed to support it. Among the more salient features of the new instructional computing environment are the user's freedom of access to a computer and the architect's opportunity to develop courseware which spans the curriculum. These features suggest that major design criteria to be optimized are the transportability and multifunctionality of the courseware. The materials should be transportable in the sense that they should be usable wherever the student has a machine to use, not just in an instructional laboratory. The courseware should be multifunctional in that, in one mode or another, it should be usable in several ways or in more than one course (e.g., as a demonstration in introductory psychology and as an experiment simulator in one or more upper-level courses).

This new perspective also emphasizes the importance of a particular role for the computer. It leads to reconceptualization of certain types of courseware. It suggests the possibility for emergence of new instructional goals for courseware. In order to highlight the similarities and differences between the old and new perspectives, it will be useful to consider some recent analyses of how computers might be used in instructional computing in ways other than having them do numerical calculations when the desired end result is the numerical calculation itself.

Instructional Computing Taxonomies

Taylor (1980) distinguished between three major "modes" of instructional use for computers: the computer as tutor, as tool, and as tutee. Castellan (1983) identified four strategies for instructional computing which he feels are particularly appropriate for use in teaching the behavioral and social sciences. These strategies are drill and practice, data generation, experiment simulation, and gaming simulation. Collyer (1984) proposed five models of instructional computer use for "things that seem to work," in fostering thinking and understanding: having the student program the computer, using the computer to replace things that already work, simulating specialized procedures and situations, providing behavior to be observed and described, and posing "problems of analysis and solution."

Taylor's modes. In Taylor's (1980) taxonomy, when the computer functions as tutor, it has been programmed by one or more experts in programming and in the subject matter of the tutorial. When the experts have done their job well, the tutor can easily adapt to a wide range of individual differences among learners. However, this mode typically requires many hours of expert work to develop a single hour of good tutoring. Among the reasons Taylor offered for the high work-to-product ratio are: (1) that humans, having intuition, are more flexible than any machine; (2) that a human tutor can omit many details of preparation, being able to count upon spontaneous improvisation to fill in both strategy and substance; (3) that programming is still an awkward, time-consuming activity; and (4) that with the computer as tutor, the stu-

dent is getting individualized lesson time which greatly increases the substantive and strategic details necessary to individualize the lesson.

The computer becomes a tool when it has some useful application capability programmed into it. The best example of this is word processing, which can be used in a variety of subjects. Other applications identified by Taylor include: (1) calculator in math and science assignments, (2) map-making tool in geography, (3) performer in music, and (4) text editor and copyist in English.

The computer assumes the role of tutor when the user must instruct the computer in the performance of some task using a language the computer "understands." Taylor argued that this is the instructional computing mode which offers the greatest long-range educational benefits. His reasons for this are threefold. First, the human tutor must understand the task before it can be taught to the computer. Second, through work with computer logic, the human tutor will learn about the workings of computers and about the working of his or her own mind. Third, no time or money is spent on finding and/or acquiring predesigned tutor software.

Castellan's strategies. The first of four strategies for instructional computing identified by Castellan (1983) is drill and practice. The most useful areas of application for drill and practice, Castellan argued, are to be found where there is a need for the learning of basic skills which can be polished by repetition with a large number of examples and where the material can be presented in small chunks (e.g., the algebraic skills needed for work in statistics courses). Optimally, the student understands, or at least has been exposed to, the general principles or theory and needs to improve skills. The advantages of computerized drill and practice lie in the immediacy of feedback, the opportunity to test understanding by working more of the same kind of problems, and the larger variety of problems that can be provided. (In other words, the student can customize the drill and practice problem set, working only with the type of problems needed and with as many as needed.)

Typically, data generators, as described by Castellan, allow the user some degree of freedom in designing an experiment, which the computer then runs by randomly generating a set of artificial data. These data are constrained by a model (e.g., the general linear model) with specified parameters derived from the results of various studies reported in the literature. After the data have been generated, the user is expected to analyze and interpret the data. Usually, the user needs to have done some background preparation in the content material before using the data generator. Also, data generators often have scoring systems set up to encourage thoughtful experimental design over simple exhaustive testing of alternative possibilities.

The third strategy identified by Castellan is the use of experiment simulators. Experiment simulators enable the user to be the subject in an experiment, with the computer both simulating the experimental apparatus and con-

trolling the sequence of events. Many programs allow the user to vary parameters and to run other "subjects" as well. Providing performance data in either raw or summary form enables the user to analyze and interpret the findings.

In discussing gaming simulations, Castellan identified the use of an abstract program or "game" to teach a set of basic principles of some kind. Castellan's view seems to be that gaming simulations are most valuable for a user who is developing skills by actively engaging in an abstract task which is an analog of situations for which prior experience might be misleading or might interfere with skill development.

Collyer's models. One of Collyer's (1984) "things that seem to work" in fostering thinking and understanding is programming. Collyer described two major advantages in learning to program the computer. The first of these is the learning that takes place about the computer's intended task during the ongoing problem analysis while the programmer is trying to get the program to work. The second kind of learning is the development of a sense of control and mastery that goes with successful accomplishment of a task.

Another of Collyer's models of instructional computing involves replacing things that already work. Collyer reviewed some suggestions for using the computer as a general-purpose machine to simulate special-purpose equipment, such as a tachistoscope, pursuit rotor, reaction timer; an IQ, aptitude, or projective test; a Stroop demonstration; or an impossible-figures display. The goal is to have one computer do the work of several special-purpose devices where the simulation works at least as well as does the special-purpose equipment.

Collyer's third model involves using the general-purpose computer to simulate specialized procedures and situations including various well-known experimental procedures. He argued that these simulations can make concepts more concrete and tractable, while providing a hands-on feel for the data-gathering process. Although simulations can be used as lab projects or in independent work, Collyer commented that the major benefits of computers as simulators may well be best exploited when they are used as parts of lectures and discussions.

In describing the model of providing behavior to be observed and described, Collyer (1984) referred to the concept of *unknown process analysis* (Collyer, 1983). This model involves having the students observe and describe the relationship which results from an unknown process's intervening between the pattern of inputs which they provide and the outputs returned by the computer.

Collyer's (1984) final model of instructional computing involves posing "problems of analysis and solution." With an unknown process governing the relationship between inputs and outputs, it is possible to engage in the kind of problem solving in which the student induces a hypothesis about the underlying process and develops procedures for conducting empirical tests which will yield confirming or disconfirming evidence. This hypothesis

testing can be done either where there is a critical test which allows evaluation of competing alternative hypotheses, or where a pattern or series of observations is required to choose among alternative hypotheses.

A synthesis. Despite dealing with the same subject—instructional computing—each of the three taxonomies described above differs substantially from the others. Although these differences could be the result of error, it seems more likely that each approach captures essential components of instructional computing. One way of reconciling the apparent discrepancies lies in recognizing that there are actually three participants in instructional computing: student, instructor, and computer (or program). Taylor's focus is upon the *roles* of the student and the program. Castellan's focus is upon *types* of programs. Collyer's focus, ignoring the activity of programming, is upon the instructor's *goals*. (Programming is expected here, because it is an activity which is done in pursuit of one or more goals rather than serving as a goal in and of itself.)

If each of these three taxonomies is treated as being a single dimension of a larger, three-dimensional categorization, we might have something that looks much like the scheme in Figure 1. Along one axis appear the three different roles for the computer; along a second, the types of programs; and along the third, the instructor's goals. While some cells in this structure may be empty, it seems reasonable to assume that most, if not all, cells could be filled. For example, an experiment simulator could be developed to tutor a student, a student could use an experiment simulator as a tool in gathering data, and a student could "teach" the computer to be an experiment simulator. In any one of these three cases, the instructor's goal might be to replace a piece of apparatus that already works, to simulate a specialized procedure, to provide behavior for observation and description, or to pose a problem for analysis and synthesis.

A New Perspective on Instructional Computing

If one believes that learning is enhanced by having the learner actively engaged in working with the raw material

out of which knowledge is constructed, two roles for student interaction with the computer—using the computer as a tool or as a tutee—are more useful than using the machine as a tutor. As with programming, the use of the computer as a tool actively engages the student in working with the raw material out of which knowledge is constructed. However, unlike programming, it does not require the student to spend enormous amounts of time in activities which contribute only indirectly to the development of knowledge.

Similarly, if one believes learning is enhanced by having the learner focus on the thinking involved in the development of knowledge rather than upon the details of procedures, two roles for student interaction with the computer—using the computer as a tutor or as a tool—are more useful than using the machine as a tutee. As with the computer as a tutor, the role of the computer as a tool focuses the student's attention upon the raw material of the structure of knowledge rather than upon the procedures used to obtain it or develop it. However, unlike the computer as a tutor, it requires the student to be actively engaged in creating that structure. In other words, the primary role for the computer in instructional computing is that of the computer as a tool, a tool with a useful application programmed into it. Because of freedom of access to computers, and the pressures it creates to have transportable and multifunctional software, it is here that the major thrust in instructional computing will occur in machine-rich environments.

Why deemphasize the role of the computer as a tutor? Using the computer as a tutor—a situation in which the student is typically in a passive role relative to the machine—will probably continue to be a part of instructional computing for a long time. It will, however, increasingly represent a smaller and smaller percentage of instructional computing at the university level, if only because students who have other, more interesting things to do on their computers will not use it, given a choice. In addition, computer-based tutorials tend to be difficult to create because they require so much attention to dialog and to anticipating all possible responses. As Taylor (1980) pointed out, the amount of time and money that must be invested in developing tutorial materials offering the benefits of both quality content coverage and individualized lessons tends to be large. This cost imposes a major selection criterion on the choice of content materials to be used as a focus for computer tutorials. It is hardly worth the cost and effort to develop a flexible, interactive tutorial on material which may be outdated within a year or so after its completion. In the long run, the type of materials that will continue to be the focus of computer-based tutorials will be those for which the content areas are relatively well structured and stable over time and/or for which there is a consistent and ongoing need for an environment in which to practice basic skills.

Why deemphasize the role of the computer as a tutee? Collyer (1984) and Taylor (1980) suggested that programming is a useful activity because of the benefits

Figure 1. A three-dimensional taxonomy of courseware.

that accrue from the discipline and the understanding of a problem area required when one has to “teach” something to the computer. (In other words, programming requires an active, structured engagement with the raw material out of which knowledge is built.) In fact, Taylor went so far as to suggest that this instructional use of the computer offers the greatest potential long-range benefit.

Programming, in and of itself, is a useful exercise. It probably does have the benefits claimed for it as an aid in understanding mathematical abstractions and the concepts embodied in a programming language (Papert, 1980). It may even be a useful thing to do if one believes that programming exercises and understanding of computer logic generalize to other worthwhile areas of endeavor. However, for many content areas, the argument that programming is an aid in understanding content is similar to the argument that using a machine tool to make a hammer is an aid in understanding how to use a hammer. It may be true, but it seems a rather cumbersome method for learning to use a hammer.

As anyone who has tried can attest, programming can be a very laborious, awkward, and time-consuming activity which also requires a lot of attention to details not relevant to the solution of content problems. Furthermore, the level of programming sophistication required to work productively with substantive content problems in an upper-level course, for instance cognitive psychology, is relatively high. Consequently, it is reasonable to wonder whether there are other ways to achieve the same results with less pain on the part of the student.

If faculty do not have to become programmers to be courseware architects, students do not have to become programmers to learn. The learning that takes place in trying to “teach” something to the computer can be accomplished in ways that do not require imposing the limitations of programming. Furthermore, the sense of control and mastery over the machine which programming may provide can be developed in a single programming course or through the use of powerful application programs, such as a word processor or a spreadsheet.

Thus, although we can use programming exercises in a few courses and may want to use them as optional projects in others, we simply do not want to turn all of our courses into programming courses. We do not want to do it for exactly the same reason that we do not want to require students to perform statistical analyses in every course they take. They may learn statistics, but the course content gets lost. Consequently, having students program the computer will not be, and should not be, of major importance as an instructional use of the computer and should constitute a relatively small portion of what happens in instructional computing across the curriculum.

A New Philosophy of Instructional Computing

One consequence of emphasizing the role of the student as an active user of the computer as a tool is the change in focus it forces upon the nature of useful course-

ware and its development. When courseware is viewed as a set of application tools which enable the student to actively create the structure of knowledge, the courseware architect's role is to design courseware so that the student is able to explore the raw material from which that knowledge is to be constructed. The student should have the freedom to make mistakes, but should not be overwhelmed by factors which may appear to be relevant, but are not. In other words, the courseware architect should seek to provide a constrained problem or knowledge space for exploration and a tool with which to explore it. One of the fringe benefits of this approach is that most software tools turn out to be multifunctional and can often do double or triple duty.

In the development of application courseware that converts the computer into a useful tool, the unidimensional taxonomies developed by Castellan (1983) and by Collyer (1984) provide a useful starting point for creating guidelines to structure the task of the courseware architect. Both taxonomies need to be refined or extended, however. For example, the goals identified by Collyer—replacing things that already work, simulating specialized procedures and situations, providing behaviors to be observed and described, and posing problems for analysis and solution—can be extended to include at least two more: developing new things that work, and creating specialized procedures and situations. Similarly, there are refinements which need to be made in Castellan's taxonomy of programs. First, the nature of drill and practice needs to be reconceptualized. Second, gaming simulations are actually a major subset of a larger category of simulations.

There is also an additional dimension which can be added to characterize an important aspect of courseware. This dimension is related to the source of the content. The content of various types of courseware can be rooted, in varying degrees, either in real situations or characteristics of the world, or in the hypothetical characteristics of an imaginary world.

Practice and exploration. Reconceptualization of drill and practice appears to be necessitated by the new directions in which traditional forms of drill and practice can be extended when every student has a computer and the machine is viewed as a tool. When the primary purpose of the courseware architect is to create a tool for exploration of a knowledge space, some programs may well contain elements of drill and practice. But, the idea of drill and practice becomes subordinated to that of repeated exploration of problems arising out of the knowledge space. Thus, the emphasis is upon practice and exploration of the nature and structure of the knowledge domain to which the program provides access, rather than on drill and practice of basic skills. (For examples, see Perkey, 1986.)

Microworld simulations. Gaming simulations belong to a category of simulations which create a microworld where the computer serves as the arbiter and enforcer of rules governing the operation of the microworld. A player-user enters the microworld and is then expected to dis-

cover some or all of the rules, to learn how to live by them, and to exit from the microworld successfully, with success being determined by some kind of scoring system under which points are awarded. Multiuser microworld simulations often involve different players in different roles requiring cooperation, competition, or some mixture of both. In single-player microworld simulations, the computer may play dual roles, serving both as arbiter of the rules and as either an opponent or a collaborator for the player.

The simulated microworld can involve re-creation of some segment of the real world, as with a flight simulator, or it can be entirely imaginary, as with some of the popular computer-based adventure games. In either case, the successful player must learn a great deal about the operation of certain pieces of special-purpose equipment that are part of the simulation. Similarly, the successful player must develop an understanding of certain specialized procedures and situations represented in the simulation. In addition, the successful player must observe and describe a number of behaviors and their consequences. And finally, the successful player must analyze and solve a variety of complex problems that require hypothesis testing and confirmation procedures.

Real versus imaginary. The discussion of microworld simulations suggests an additional dimension to courseware. This is the dimension of real versus imaginary sources for the instructional content. Practice and exploration exercises can be used to work with basic knowledge to which the student has already been exposed, and they can be used in the development of new knowledge structures which might be useful or required in some possible future world. Data generators can be constrained by models having parameters derived from real data or from the characteristics of the model, or they can be constrained by the characteristics of a model of some imaginary situation with interesting or useful characteristics. Experiment simulators can be used to recreate and replicate "classic" studies with the student as subject, as experimenter, or both, and they can be used in the creation of experimental conditions or procedures which have not been used before or which may be useful in some possible future situation. Finally, microworld simulations can be used to simulate the operation of real-world processes, or they can be used to create an imaginary microworld having some particularly useful, desirable or simplified characteristics.

Summary: A new taxonomy. Pulling together the various threads running through the analysis of courseware described above, the overall result is a view of a courseware development which emphasizes the desirability of viewing the computer as a tool to accomplish certain goals using one of several types of programs which may contain instructional content from either real or imaginary sources. This analysis is illustrated by the three-dimensional taxonomy of courseware for the computer as a tool represented in Figure 2. This figure differs from Figure 1 in that two layers of cells, those for the com-

Figure 2. A courseware taxonomy for the computer as a tool.

puter as a tutor and those for the computer as a tutee, have been stripped away, leaving behind only those cells representing uses of the computer as a tool. Another dimension, the source of the instructional material, has been added to the figure, and the refinements and new categories described above have been included.

As implied by Figure 2 and the discussion above, in a machine-rich environment, there are several types of programs—practice and exploration, data generators, experiment simulators, and microworld simulations—which can be developed as tools. Any of these tools can be shaped to serve one or more of several goals: to replace things that already work, to develop new things that work, to simulate specialized procedures and situations, to create specialized procedures and situations, to provide behavior to be observed and described, and to pose problems for analysis and solution. The instructional content of the courseware tools can come from either of two sources. It may be based directly upon considerations and factors existing in the real world, or it may be derived from some imaginary world with useful or interesting characteristics. Bringing to bear a thorough knowledge of the content area, it is the role of the courseware architect to determine the goal(s) that the courseware is to serve and to select the type of program which gives the best fit with the goal(s) and the knowledge structure or problem space—real or imaginary—that the student is being asked to explore.

Courseware for the Psychology Curriculum

At the time the microcomputer project was begun at Drexel, there was no Macintosh courseware of any kind for any psychology course. To deal with the problem of developing courseware for the psychology curriculum, our department has been working on a two-part solution. The first part involves the conversion of a set of previously class-tested materials for use in the new context. The second part of this approach is directed toward the develop-

ment of a systematic process for identifying and developing a curricular resource catalog of simulations, experiments, and so forth to be targeted for future development.

Short-range strategy. Working with instructional support programmers, Chute (1986) has been the architect involved in converting a series of class-tested experiments and demonstrations into a set of Macintosh-based experiments. The sequence of experiments and exercises from which Chute has been selecting his materials are part of a frequently utilized set developed over an extended period and class tested in a traditional format. These experiments are used to teach students a variety of fundamental concepts in general experimental psychology.

Chute's long and active involvement in the refinement of these materials, and his frequent prior use of them in an instructional environment, has helped to ensure the initial selection of a pedagogically useful subset for computerization. Now that the first set has been computerized, it is undergoing formative evaluation as members of the faculty utilize it as part of the instructional materials in the introductory psychology course. Meanwhile, additional architectural specifications and courseware modules are under development.

Long-range strategy. The second part of our attack on the problem of developing courseware for the psychology curriculum is to develop a resource catalog of simulations, experiments, demonstrations, and so forth which can be implemented in computer-based form. The long-range goal is creation of a library of Macintosh-based courseware modules from which individual faculty members will be able to select when structuring their courses at both the introductory and advanced levels. Many of these modules will be usable wherever the student has a machine.

CONCLUSION

When every student has a personal computer, the widespread availability of this new technology creates a new challenge and a new opportunity. What is unique is the opportunity to widely disseminate the raw material out of which knowledge is built, and the tools with which to build it. The challenge is to create courseware which captures that knowledge and its structure, and which enables the student, using the machine as a tool, to find his or her own way through that knowledge space.

For example, machine-as-tool courseware application programs, such as experiment simulators, are of special

value when the user is more interested in the thinking behind an experiment than in the experiment itself. By asking students intelligent questions, one can get them to play with simulations and discover the underlying processes for themselves. This is especially effective when each student has a computer. It allows all the time a student needs for changing various parameters to answer, "What if...?" Thus, this kind of courseware enables a student to experience directly the stuff from which knowledge is derived and to learn to work with complex knowledge intuitively, and in so doing get a feel for how different changes affect a system.

The machine-rich environment also brings us closer to the change in the nature of the instructor's role which has been forecast since the early years of computer-assisted instruction. Increasingly, courseware will consist of special-purpose applications programs which the student uses to explore knowledge space within the constraints of the courseware application tool. This kind of courseware and the new instructional computing environment enables finding one's own way through knowledge. Increasingly, the role of the instructor will focus more on the process of motivating and guiding students' explorations of the structure of the knowledge contained in the application, or accessible through its use, and less upon transmitting information.

REFERENCES

- BROOKS, F. P., JR. (1975). *The mythical man-month*. Reading, MA: Addison-Wesley.
- CASTELLAN, N. J., JR. (1983). Strategies for instructional computing. *Behavior Research Methods & Instrumentation*, **15**, 270-279.
- CHUTE, D. L. (1986). MacLaboratory for psychology: General experimental psychology with Apple's Macintosh. *Behavior Research Methods, Instruments, & Computers*, **19**, 205-209.
- COLLYER, C. E. (1983). An introduction to unknown process-analysis, with a dialogue. *Teaching of Psychology*, **10**, 88-93.
- COLLYER, C. E. (1984). Using computers in the teaching of psychology: Five things that seem to work. *Teaching of Psychology*, **11**, 206-209.
- HEWETT, T. T. (1985). Teaching students to model neural circuits and neural networks using an electronic spreadsheet simulator. *Behavior Research Methods, Instruments, & Computers*, **17**, 339-344.
- HEWETT, T. T., & PERKEY, D. J. (1984). The mythical "mountain" of software. *Collegiate Microcomputer*, **2**, 207-210.
- PAPERT, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.
- PERKEY, M. N. (1986). The effect of a machine-rich environment on courseware development: The process and the product. *Behavior Research Methods, Instruments, & Computers*, **19**, 196-204.
- TAYLOR, R. P. (1980). *The computer in the school: Tutor, tool, tutee*. New York: Teachers' College Press, Columbia University.