*Computers in Human Behavior,* Vol. 2, pp. 235-238, 1986
Printed in the U.S.A. All rights reserved.

0747-5632/86 $3.00 + .00
Copyright © 1987 Pergamon Journals Inc.

# Book Review

**Computer-Based Instruction: Methods and Development.** S. M. Alessi and S. R. Trollip. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1985, xiii + 418 pp., $25.95, paperback.

The history of computers has been traced to various origins, but ENIAC (Electronic Numerical Integrator and Computer) is a good candidate for the distinction of first electronic, general purpose computer. When it was activated in February of 1946, it occupied 15,000 square feet of floor space and had to be rewired literally every time it was to perform a new function. About twenty years later, IBM touted the "360" as its flagship mainframe. It featured 256K "core memory" and cost the modern equivalent of $800,000. Another 20 years has put computers with more power and more versatility on desktops at prices below $2,000.

The continuing evolution of "smaller and better" computer technology is the stuff of science fiction and is enough to make anyone wonder where it will lead. Corporations, universities, and school districts are particularly interested in the future of classroom computing and many are betting that it will prove to be more fact than fiction. Apple, Commodore, Tandy/Radio Shack, and IBM are investing great efforts to capture parts of the educational market estimated to be worth more than $2.3 billion dollars over the next four years in hardware alone (Reinhold, 1986).

Despite the willingness of some to make these large investments, others suspect the computer will have no more success in the schools than did other much-heralded technological innovations, such as, the phonograph, television, and teaching machines. Though the phonograph and television have found wide acceptance in American society, they have never been widely used devices for school instruction. Recent surveys have shown that schools will frequently teach *about* computers, but less frequently teach *with* computers (Becker, H. J., 1983; Corporation for Public Broadcasting, 1986).

Perhaps the greatest threat to the acceptance of computer-based instruction is the lack of adequate software. An assessment project which surveyed hundreds of programs recommended only 30% to 40% for use and found only 5% to be of truly high quality. Only 20% of the programs had been pilot tested with learners during development (Komoski, 1984, p. 247). Hardware can only be as good as its software. In the educational arena, new fleets of powerful microcomputers look like fine ships sent to sea without pilots.

It is in this context that the book *Computer-Based Instruction: Methods and Development* is presented. The authors express their legitimate concern over the poor quality of typical instructional software and offer detailed advice from their own experience with program development as a contribution to the necessary remediation. In three sections, the authors touch on all the major themes relevant to computer-based instruction: hardware, instructional design, and software development. The meat of the book is in the second section and it consists of meticuluous descriptions of and prescriptions for effective tutorials, drills, simulations, instructional games, and computer-based testing. The first section gives cursory sketches of the computer's history in and out of the classroom and of hardware and software that developers should know about. The third section concerns the processes related to software development, such as, flowcharting and programming. One good feature of this section (and the whole book) is that it is not machine- or language-bound. The recommendations are generic and do not presume a familiarity with any particular computer or programming language.

The strategy of empowering novice developers with explicit rules for the creation of effective software is an appealing one. Local software development could be more responsive to the specific needs of a particular school district, school, class, or even student. Programming instruction could clarify the content and methods of instruction for teachers, parents, and students, and could help in isolating especially effective and especially weak components of a course or a curriculum. Finally, the same standards that *Computer-Based Instruction* recommends for developers could be borrowed by users and purchasers of software to sift the wheat from the chaff.

What should developers and evaluators attend to if they are to improve the quality of instructional software? And does *Computer-Based Instruction* provide the necessary direction? I think developers and, perhaps even more, evaluators of instructional software would do well to examine this book. Its consideration of different software types is thorough and detailed and even experienced programmers will probably be led to consider software features that they had inadvertently neglected in the past. The book's great virtues are its thoroughness and detail; its great drawbacks are that it seems somewhat dated and conservative in its conception of software development.

Section 1, "Computers and Their Applications," is probably the weakest part of the book. The historical background and discussion of differences in computer hardware is superficial and has the quality of being an afterthought. In the discussion of hardware, Apple's discontinued Lisa computer is pictured, as is a discontinued Texas Instrument's home computer. It is unclear whether the authors purposefully chose these representatives or publication delays rendered some of this section obsolete. Though these discontinued models are included, no mention is made of Apple's Macintosh. The Macintosh's uniquely user-friendly combination of menu-driven, icon-based, mouse-controlled programs complete with windowing functions makes it a computer that simply cannot be ignored in a book discussing instructional computing. Mice and Macintoshes are not the only items missing from the discussion of hardware. Modems, conferencing and networking, hard disks, and other devices never made it to the text.

History and hardware are tangential to the main purpose of this book. But other sections of the book are weakened when they do not elaborate on the relationship between development issues and hardware. For example, under what educational conditions would a joystick be preferred to, say, a light pen? Nor does the book consider novel integrations of new and old instructional systems. For example, how could a "traditional" computer-based tutorial be enhanced by allowing computer conferencing while the lesson is in progress?

Section 2, "Methods of Computer-Based Instruction," classifies instructional software into five groups: drills, tutorials, simulations, instructional games, and testing programs. These seem to be fair and well-defined distinctions. The authors rightfully qualify their taxonomy, suggesting that some software will combine elements of more than one of these types, such as, a tutorial interspersed with short simulations. Unfortunately, the book does not examine how such combinations could work and I suspect the average reader will finish the book conceiving of software in just five categories.

The second section is eminently practical, often suggesting what to do without explaining why it should be done this way. Though this approach will help developers get into production right away, the dearth of theory may be a mistake for the long haul. When a programmer is faced with a novel problem or wishes to set out on some untraditional course, he or she would still want to ground a new program in some set of principles that are presumed to be true of instruction and learning generally. Such principles are rarely discussed in the book and, when they are, they are often not related to the specific recommendations offered.

That's not to say that the book has no guiding models. The ones that are given seem highly indebted to the behavioristic tradition. The instructional model, for example, borrows four of nine instructional events proposed by Gagné (Gagné, 1970; Gagné, Wager, & Rojas, 1981). The authors believe all good instruction must present information, guide the student, provide opportunities for practice, and assess student learning. These are, of course, alternative descriptors for a shaping process using cycles of stimulus-response-reinforcement.

Gagné's instructional events are a good place for a designer to start, but they are not without their shortcomings. First, they are vague and often cannot give clear direction. To say that instruction means providing information, guiding the student, etc., almost begs the question. Now we must ask what it means to provide information, guide the student, etc. Also, the book's model implies a lock-step approach to instruction, that one must first present information, then guide students, then provide practice, etc. Under some circumstances, it may be desirable to change the order of events. In fact, in some circumstances, it may be desirable to withhold some of the events. It may be instructionally sound, for example, depending on the students and the content to be learned, to *not* present information but require students to discover it for themselves.

In section three, *Computer-Based Instruction* introduces a model of software development that would be useful for novice developers and those who are dissatisfied with their current methods. Like the instructional model, the development model has a behavioristic flavor. Its eight steps can be described in three parts: the organization of ideas, production of instructional materials, and evaluation of the product. Two comments should be made about the model.

The first part of the development model, the organization of ideas, is very similar to methods for the development of behavioral objectives. The authors do well to explicitly direct developers to review the available resources, including previous programs in the area, and to use brainstorming techniques to generate ideas for the instructional program. The authors also discuss "conceptual analysis" as an alternative or supplement to "task analysis."

Programming, and instruction generally, would improve if all teachers would be so explicit in creating their lessons. But what's missing from this approach is attention to the *cognitive processes* that learners can be expected to engage in when presented with particular elements of programs. Another way of designing instruction then would mean beginning with clear statements of the kinds of mental processes, strategies, skills, and procedures we wish our students to acquire during our instruction. If we wish to only convey information, we could do the information processing for the students, thereby "short-circuiting" their own mental activity. Unfortunately, most instructional software is of this "short-circuiting" type. If we think that our students either already have the skills we want to teach and simply need to refine them or are gifted enough to invent them on their own, we can provide "activating" tasks, complex tasks that require students to exercise the skills we wish to teach. Finally, we can use the computer to "model" mental activities for the student, giving them a clear example of how particular mental strategies can help resolve difficult problems (Corno & Snow, 1986). I hope that consideration of "short-circuiting," "modeling," and "activating" mental strategies becomes as much a part of instructional design as performance and content analyses should also be.

The second part of the development model described in *Computer-Based Instruction* involves the production of instructional materials. This section is good, though one caution should be added. The model advocates the use of storyboards, separate sheets of paper representing individual displays that learners will see. Though this would be useful for simple programs, storyboarding will almost inevitably result in programs resembling what former Secretary of Education T. H. Bell called "electronic page-turning," and this is especially true for novice developers. Granted, the computer will be able to "turn pages" quickly and in complex ways, but we should be able to expect more from such sophisticated devices. Storyboarding can be especially cumbersome if not impossible to use when the program involves complex animation, or when users have control over the appearance of the screen, as in the multiple window capacity of the Apple Macintosh.

For a book that is meant to help software developers and users improve on the poor showing of instructional programming to date, it is strange that the authors do not expound on how the efforts to date have failed. To change the way they work, designers and programmers need both an ideal to pursue and a clear conception of what to avoid, of what has been tried and what needs to be improved. Bork (1984) gives a helpful list of 13 factors that characterize poor instructional software. These include "failure to make use of the interactive capabilities of the computer, failure to make use of the capabilities of the computer to individualize instruction, too-heavy reliance on text, treatment of the computer screen as though it were a book page, content that does not fit anywhere in the curriculum, and use of long sets of instructions at the beginning of programs that are difficult to follow and difficult to recall" (p. 241). A list like this would be a good place to *start* a book on improving instructional software, and I'm afraid that the present book does not examine these problems in much detail.

For developers and evaluators of instructional software, Alessi and Trollip's book *Computer-Based Instruction* is a good book to have on hand. It gives a handy reference to criteria for evaluating instructional software and is full of tips for software development, the kind of tips that one discovers only after years of work in the area. But readers are also advised of its limitations. It is short on theory, a little outdated in places (something that is almost unavoidable in such a fast-changing field), and somewhat conservative in its conception of computer-based instruction.

Robert L. Bangert-Drowns
Center for Research on Learning and Teaching
University of Michigan
Ann Arbor, MI

## REFERENCES

Becker, H.J. (1983). *School uses of microcomputers, Issue no. 1.* Baltimore, MD: Center for Social Organization of Schools, The Johns Hopkins University.

Bork, A. (1984). Computers in education today—And some possible futures. *Phi Delta Kappan*, **66**, 239–243.

Corno, L., & Snow, R.E. (1986). Adapting teaching to individual differences among learners. In M. Wittrock (Ed.), *Handbook of research on teaching.* New York: Macmillan.

Corporation for Public Broadcasting. (1986). A national study of the educational uses of telecommunications technology in America's colleges and universities. *Research Notes, No. 21.* Washington, DC: Author.

Gagné, R.M. (1970). *The conditions of learning.* New York: Holt, Rinehart, and Winston.

Gagné, R.M., Wager, W., & Rojas, A. (1981). Planning and authoring computer-assisted instruction lessons. *Educational Technology*, **21**, 17–21.

Komoski, P.K. (1984). Educational computing: The burden of insuring quality. *Phi Delta Kappan*, **66**, 244–248.

Reinhold, F. (1986). Micro marketers: How hardware vendors plan to woo schools. *Electronic Learning*, **6**, 30–36, 83–84.