Instructional Design and Authoring Software

Greg Kearsley
Courseware, Inc.
20075 Carroll Canyon Road
San Diego, CA 92131

Abstract. This article explores the instructional design aspects of authoring software used to develop Computer Based Instruction (CBI) programs. The significance of following good design principles for interactive instruction is emphasized. The nature of authoring languages and authoring systems is discussed. A series of practical considerations in using authoring software and implications for instructional design are described. Finally, needed developments in authoring software including intelligent tutoring and automated instructional design systems are outlined.

With the increasing use of computer based instruction for a wide variety of education and training applications, a great deal of attention is now being focused on the design and development of instructional software (e.g., see Kearsley, 1984; Walker & Hess, 1984). One area of particular attention is authoring software, i.e., programs which helped authors create computer based instruction in a timely and cost-effective manner. The relationship between instructional design and authoring software is explored in this article.

Design of Interactive Instruction

Before discussing the characteristics of authoring software, some remarks about the design of interactive instruction are in order. Since a number of good texts have now been written on this topic (e.g., Allessi & Trollip, 1985; Dennis & Kansky, 1984; Steinberg, 1984) extensive discussion is unnecessary here. However, it is critical to understand that such design principles exist and directly affect the quality and effectiveness of instructional programs. For example, consider the ingredients of good screen design (e.g., Grimm, 1983; Heines, 1983; Jenkin, 1981). Good screens are not crowded, avoid full screen scrolling, organize information functionally, are titled, use color and graphics effectively, focus the attention of the learner, and use windows or screen partitioning to present concurrent information.

Conversely, poorly designed screens are too crammed with information, scroll, lack titles, have information located randomly, do not use color or graphics, fail to focus attention, and attempt to present concurrent information in a sequential fashion.

The creation of good screen displays depends upon knowledge and understanding of these principles, not the particular authoring software used. Quality CBI results from following good design principles, not the use of particular authoring software. While this point may seem obvious, a great deal of time is sometimes spent debating the best way to implement an instructional program while neglecting the design principles that really make the difference.

This is especially true of the pedagogical frameworks that usually underly the development of authoring programs. For example, the TICCIT System was based upon a model of componentized learner control that characterized all instruction developed for that system (Bunderson, 1974). Gagne, Wager and Rojas (1981) outline a model for developing programs based upon Gagne's instructional theory. Bork (1984) describes a methodology which prescribes different contributions made by various team members. These design and development frameworks affect the nature of the instructional programs produced much more than the particular authoring software used.

Authoring Computer Based Instruction (CBI)

The range of authoring software available today is quite impressive. Historically, authoring programs were developed for dedicated CBI systems (such as PLATO and TICCIT) or IBM mainframe computers (such as IIS, Phoenix, or Scholar/Teach). The evolution and characteristics of these early authoring systems are discussed in Kearsley (1982). With the emergence of microcomputers, dozens of authoring programs for popular computers have become commercially available (see Kearsley, 1984). Furthermore, a number of special purpose authoring programs (e.g., for creating graphics, interactive video or simulations) have also appeared.

Authoring Languages and Authoring Systems

An important distinction necessary to understand authoring software is the difference between an authoring system and an authoring language. An **authoring language** is a special purpose programming language designed solely for creating instructional programs. An **authoring system** is a high level interface that allows an author to generate an instructional program without any explicit programming, simply by specifying the instructional content and teaching logic. The distinction between an authoring language and an authoring system is important because it reveals a fundamental characteristic of authoring software. Both authoring language and authoring systems provide structure to the instructional design process. An authoring language provides simplified programming commands for creating screen displays and graphics, for answer analysis, and for branching. An authoring system goes one step further and provides an actual lesson framework, complete with an implicit or explicit pedagogical strategy. It is this structure of an authoring language or authoring

system that makes it possible to create instructional programs faster and with less effort than using a general purpose programming language (such as BASIC or Pascal).

However, it is also the same structure that imposes the major limitations of authoring software. As long as the pedagogy built into the authoring language or authoring system is appropriate to the application (or acceptable to the author), then the tool is useful. But, if the structure limits what the author would like to do, the authoring software becomes a problem and constrains the development process and the quality of the instructional programs created. Hence, the dilemma for the developers of authoring software: how to design authoring tools that provide maximum flexibility in the creation of instructional programs, while also providing sufficient structure to minimize the time and effort required in the authoring process.

Authoring Languages

There are many authoring languages available for both dedicated CBI systems (e.g., PLATO, TICCIT) and personal computers. Three authoring languages in particular have strongly influenced the CBI field: Coursewriter, Tutor, and Pilot. Coursewriter was originally developed by IBM for their 1500 Instructional System and then became the basis for the Interactive Instructional System (IIS) which runs on IBM mainframe computers. IIS is widely used in the training world, especially for applications involving data processing or administrative training.

Tutor is the authoring language developed for the PLATO system. Tutor is probably the most complex authoring language in existence with hundreds of commands and parameters. It has been used to create thousands of PLATO programs for a diverse range of educational and training applications. A modern derivative called TenCore is now available for IBM personal computers.

Pilot was specifically designed as a simple authoring language that could be used by teachers. One of the things that makes Pilot easy to learn is a relatively small number of commands (about 20). Pilot has the distinction of running on most microcomputers and hence comes closest to allowing transportability of instructional programs from one computer to another. Because Pilot is available for most microcomputers, it is also the best known but not the most used authoring language. All three of these authoring languages exhibit similar features. This includes commands to:

- display text on the screen
- match student responses
- assign values to variables
- branch instruction on specified conditions
- collect student performance data

The three languages differ in their capabilities to create graphics, character sets, sound, and handle non-keyboard input (e.g., touch, joystick, light pens).

Authoring languages such as Coursewriter, Tutor, or Pilot differ from general purpose programming languages (such as BASIC, Pascal, Forth, etc.) mainly in the additional capabilities provided for creating screen displays, answer analysis, and collection of student performance data. The net result of these differences is that authoring languages make it faster to create an instructional program since the features necessary or desired for instruction are present in the language.

Authoring languages provide relatively few constraints on instructional strategies or pedagogy, unlike authoring systems. For example, you can program tutorials, drills, games, simulations or tests using any authoring language. Depending upon the features of the specific authoring language, it may be easier or harder to do what you want—but, it is undoubtedly possible given sufficient programming expertise. The last remark reveals the real limiting factor with authoring languages; they are as good as the programming skill available. The development of high quality instructional programs using authoring languages is not only dependent upon good design but also, upon the availability of very experienced programmers. Should instructional designers attempt to do the programming themselves, they will need to become accomplished programmers in order to implement their designs in a sophisticated fashion.

Authoring System

An authoring system is basically a program that generates other programs. By choosing options from menus or responding to questions, the information needed to create an instructional program is provided without actually having to write the program. This means that an author does not have to learn to program or even rely on a programmer to produce lessons. The lesson content can be entered in the computer directly from storyboards or scripts without having to be translated into programming instructions.

When the system generates an instructional program, the program is free of "bugs." Since debugging a program typically account for about 50 percent of the total time spent developing software, the elimination of the debugging step is a tremendous improvement in productivity. Thus, the two major advantages of authoring systems over authoring languages are:

(1) programming skills are not required, and

(2) considerable time is saved because no debugging is needed.

As explained earlier, the primary limitation of authoring systems is that they can significantly constrain what can be done instructionally. For example, suppose that the question (or test item) component of the authoring system allows a wide range of possible question types but you want to create a type of question which is not provided. Or, suppose, you wanted to measure how long it took the student to answer the question and this was not a feature built into the authoring system. Or, perhaps you want to create a game, but there is no scoring capability available. Most instructional designers can very quickly generate ideas that cannot be accommodated by the fixed format of an authoring system.

Software does not necessarily make instruction easy to create.

There are basically three solutions to this dilemma. The first is to accept the limitations of the authoring systems and avoid designing any instruction that can't be done using the authoring system. Keeping in mind that authoring systems can substantially decrease the time and costs associated with developing lessons, this may be a desirable strategy for some applications.

The second solution is to use a combination of authoring systems and author/general purpose programming

languages. The authoring system is used as much as possible, but things that can't be done using the authoring system are programmed and incorporated into the program via an authoring or general purpose programming language. Some authoring systems include an authoring language and make it easy to go between the system and the language. This strategy allows the author to take advantage of the efficiency of the authoring system as much as possible and to have the flexibility of an authoring language when needed.

The third solution is to allow the author to modify the authoring system and add new instructional features. This is done by designing new templates which include the desired capabilities. For example, if you wanted to have multiple responses to a question, you would use the authoring system to create a new option to do this. In some authoring systems that allow this capability, all authoring programs are just lessons themselves and they can be created or modified in the same manner as any other lesson created.

Authoring software should also assist in the analysis, design, and evaluation stages.

In order to define a new authoring template, the author must be able to explicitly state the instructional strategy desired. This is often more difficult than it sounds. Unless an instructional developer has had considerable experience designing interactive instruction, the detailed specification of instructional logic required for a CBI program is a formidable task. For this reason, the creation of instructional programs is still difficult even when an authoring system is used. The authoring system eliminates the need to program but not the requirement to think logically and in procedural terms about an instructional sequences being created.

Most authoring systems do not allow the capability to make extensive changes to their pre-defined structures. As a consequence, authors must use the inherent pedagogy and instructional strategies of the authoring system. The predominant pedagogy in these systems is conventional drill, tutorial, or test programs with static graphics. Few authoring systems make it easy to create simulations or games with animated graphics. Hence, the use of currently available authoring systems does not encourage particularly innovative instructional programs. For this reason, most commercial developers of instructional software use general purpose or authoring languages instead of authoring systems.

Authoring Software Capabilities

From an instructional design perspective, the most important aspect of authoring software is the range of capabilities provided. The more functions provided, the greater flexibility possible and the wider range of instructional applications which can be created.

Table 1 provides a list of desirable capabilities for an authoring tool divided into four levels of authoring: content creation, lesson definition, course management, and authoring environment. Content creation functions include the input and modification of text, graphics, sound or synthesized speech, and variables. Content creation represents the most time-consuming part of the authoring process, but it is not the most complex. Lesson definition is the most complex aspect of authoring because it involves the assembly of displays, specifying answer analysis sequences, and defining branches. Course management functions include lesson sequencing, try-outs, specifying response data to be collected, specifying student control options, designing instructional strategies, automated documentation, message options, and debugging aids. Authoring environment functions include new template definition, specifications of hardware configurations, automatic language translation, lesson compression, and availability of helps/training for the authoring software.

As a rule of thumb, the more of these capabilities provided by the authoring software, the greater range of instructional design possible. Content creation and lesson definition capabilities determine what the display will look like and how easy or hard it is to design different types of CBI (i.e., drills, tutorials, simulations, games). Most authoring languages and authoring systems available today provide good capabilities for content creation and lesson definition, although they are often limited in terms of different type styles and graphics possible and the range of branching allowed.

However, relatively few authoring tools provide capabilities at the course management or authoring environment levels. This means that authors must take care of assembling lessons, specifying student control options or defining new strategies or templates themselves via an authoring general purpose language. In many cases authors are expected to become versatile with the particular operating system used (e.g., DOS, UNIX, CP/M) in order to perform these functions. As a consequence most authoring programs available today allow good quality instructional programs to be created within a limited range of traditional CBI strategies (such as drills, tutorials, simulations, etc.). The limitations of authoring may be either accentuated or remedied by the design expertise and sophistication of the author and by several practical considerations in the authoring process.

Practical Considerations

Practical considerations are often much more serious limitations to the authoring process than instructional design constraints. Characteristics of the delivery system tend to dictate what features are possible. Table 2 lists some major system characteristics that determine the nature of a CBI program. Clearly, the availability of certain capabilities in an authoring program will depend upon the characteristics of the intended delivery system.

Compatibility. Authoring software is usually designed to be used with a specific computers. Pilot is a notable exception. This means that the instructional programs created will only run on the specific computer used for authoring. Furthermore, the pace of innovation in the computer field is frantic. Many developers have spent considerable time creating instructional programs for machines which were no longer popular or sold by the time they finished their programs. The dilemma is that good authoring software is usually not available for a machine until a year or more after its introduction in the marketplace.

Usability. It was noted in the preceding section that authoring software becomes more powerful in an instructional sense as additional functions are added. However, the price paid for the additional power is complexity. More powerful authoring languages or

Table 1:
Authoring Software Characteristics

CONTENT CREATION

Text
- input mode (promoting, menus/fields, commands)
- formatting (margins, justifications, character size, fonts, word wrap, spacing, pagination)
- editing (line/page mode, globals, insert/replace/delete/copy)
- selective erase

Graphics
- simple line (boxes, circles, graphs)
- illustrations (drawing, digitization)
- dynamic (animation)
- character sets
- library

Sound/Speech Editor

Variables
- stored
- random

LESSON DEFINITION

Display Integration
- text, graphics, sound/speech
- multimedia (slides, video, audio)
- editing (move/delete/replace)

Student Response Processing
- response type (single character, M/C, string match, keywords, algebraic, touch/cursor position)
- answer logic (explicit/implicit, Ca/Wa/Un designation, feedback messages)

Branching
- unconditional (labels, function keys, restart)
- conditional (counters/buffers, arithmetic, logical)

Documentation (comments, objectives)

COURSE MANAGEMENT

Lesson Sequencing/Linking
Lesson Try-Out Mode
Response Data Specification
Student Control Options (back/forward, helps, maps)
Instructional Strategy Definition
Documentation
Message/Mailbox
Debugging Aids

AUTHORING ENVIRONMENT

Template Definition
Hardware Configuration Specification
Language Translation
Lesson Compression
Uploading/Downloading (Networking)
Helps/Training

authoring systems typically take much longer to learn and take longer to create lessons. This means that the time required to become proficient at authoring can take many weeks or months. Lessons developed before the author has mastered the software may tend to be ineffective and not of professional quality.

Performance characteristics. Most systems have some aspects which are slow, and authors change their design approach to accommodate these. For example, if an authoring system is slow in presenting new screens, authors tend to cram more content on each screen. If the authoring system is slow to assemble graphics, authors tend to simplify their graphics or reduce the number used. If a particular strategy is cumbersome to create, they will use another strategy. In short, the performance weaknesses of an authoring program are likely to affect the instructional design of the lessons.

Networking capabilities. If networking capabilities are available, a greater tendency exists to collaborate during the authoring process in terms of shared creation and editing responsibilities. If the authoring program only runs on a stand-alone microcomputer, it is difficult for an authoring team to work together on the same program. While this can be handled by close coordination in the specification stage, in practice, good team authoring often requires a networked system.

Needed developments

The preceding remarks have alluded to a number of areas where advances in authoring software are needed. While the point has been made that currently available authoring software will allow good instruction to be created, software does not necessarily make instruction easy to create. For example, most authoring systems provide the capabilities needed to create simulations and games. However, the present collection of authoring languages and authoring systems do not facilitate their creation. As another example, consider test item generation. Most authoring software will help you create a wide variety of tests but not help you write test item generators.

A major deficiency of current authoring software lies in the domain of intelligent tutoring systems or ICAI (Sleeman & Brown, 1982). ICAI programs use artificial intelligence programming techniques to provide a "deeper" level of understanding about the content being taught and the student's learning progress. These programs involve knowledge networks which are acted upon by tutoring rules in order to generate student dialogues or problems. They are usually "mixed initiative" in nature in that either the student or computer can take the initiative at any time to ask a question or pursue an idea.

A number of ICAI programs have been constructed in the past decade as research paradigms. These include SCHOLAR, a geography tutor (Collins, 1977), SOPHIE, a tutor for electronics troubleshooting (Brown, Burton & deKleer, 1982), WEST, a coach for game playing (Brown, Burton, 1978) and STEAMER, an intelligent simulation (Stevens & Roberts, 1983).

As a consequence of these and other ICAI projects, it is now clear how to design more sophisticated CBI programs than the ones in current use. The problem is that the construction of these programs is fundamentally different from the design and development process we presently follow. Furthermore, the authoring process for ICAI programs relies heavily on the knowledge of artificial intelligence programming language (such as LISP or Prolog). If ICAI programs are to be developed and used more widely, authoring software is needed that can be used to develop such programs easily.

The biggest deficiency of current authoring software, however, lies in its

narrowness. It was emphasized earlier in this article that the use of good design principles overshadows the importance of authoring software limitations. Most of the effort in developing instructional programs lies in the analysis, design, and evaluation stages. Therefore, if authoring software is to have an important impact on development, it should also assist in the analysis, design, and evaluation stages.

A number of research studies have been conducted in the area of automated instructional development systems. Braby and Kincaid (1981) describe a system for the design and production of technical manuals. O'Neal and O'Neal (1979) discuss a prototype authoring management system to be used in large instructional projects. Merrill and Wood (1984) describe their Lesson Design System for automating instructional design. At Courseware, Inc., a system called the Instructional Toolkit is being developed which automates the ISD process and runs on a personal computer.

This area of research should ultimately result in full-scale authoring systems which streamline the entire development process and improve the quality of instructional materials. Ideally, such systems should provide an integrated set of programs for the analysis, design, development, implementation, and evaluation of instruction of any media (not just CBI). To the extent that good instructional design principles are built into these programs, it should be possible for novice instructional designers or content experts to create high quality instructional materials. And, for instructional design experts, an automated instructional development system should be a productivity tool, increasing their efficiency.

There is a rather profound connection between these two lines of research. Work on the Instructional Toolkit has shown that many aspects of the instructional design process can be automated by using the same programming methods underlying applications such as word processing, spreadsheets, and database management. However, many of the more complex aspects of design are very intuitive and heuristic in nature. In order to automate these processes, artificial intelligence programming techniques and theory will be needed. Thus, the design of authoring software for creating intelligent CAI tends to bring together these two research areas.

Conclusions

One of the intriguing aspects of attempts to automate instruction is that it forces us to examine the process of developing instruction to a much greater depth than in the past. Authoring software has forced us to think about exactly what features are desirable in a CAI program and the process of creating an instructional program. Intelligent CAI makes us look much more closely at what it means for a student to understand a concept or a subject. The development of authoring systems to create ICAI programs will require an even greater analysis of the process of creating effective instruction. Similarly, the development of automated instructional design systems requires a very detailed study of the instructional development process. Thus, the net effect of developing authoring software is that it makes us look deeper at our instructional design methodologies. Future developments in authoring tools will extend this analysis. For this reason authoring software is of both practical and theoretical significance.

Table 2
Characteristics of Delivery System

Display Characteristics
- alphanumeric vs. graphics
- monochrome vs. color
- screen resolution

Input Characteristics
- standard keyboard
- function keys
- cursor control (arrow keys, mouse)
- touch/light pen

Performance Characteristics
- type/capacity of disk
- type/speed of processor
- local/remote network capabilities

Multimedia Capabilities
- slide/tape
- video
- speech I/O
- A/D devices

The net effect of developing authoring software is that it makes us look deeper at our instructional design methodologies. Future developments in authoring tools will extend this analysis.

It is important to realize that advances in authoring software are primarily driven by new developments in computer technology. for example, the emergence of machines like the Apple Macintosh with high resolution graphics, extensive windowing, pull-down menus, and the mouse, mean new capabilities are now possible for CAI programs and, hence, authoring software. Similarly, the emergence of interactive videodisc and its continuing evolution mean new possibilities for CAI programs and authoring software. Because authoring software is so intimately associated with the capabilities of hardware and software, it is likely to always be in a state of change.

We are moving towards an era when computer literacy will be required of any instructional designer. This will mean the ability to use authoring software of one form or another to create instructional materials. It is likely that the better designers will be those individuals with the greater mastery of authoring tools. Our current authoring software is just a crude foreshadowing of things to come. Yet even these primitive programs can already help considerably in producing better quality computer instruction.

References

Allessi, S. M., & Trollip, S. R. (1985). *Computer based instruction: Methods and development.* Englewood Cliffs, NJ: Prentice-Hall.

Bork, A. (1984). Production systems for computer based learning. In D. Walker & R. Hess (Eds.), *Instructional Software.* Belmont, CA: Wadsworth.

Braby, R., & Kincaid, J. P. (1981). Computer aided authoring and editing. *Journal of Educational Technology Systems, 10* (27), 109–124.

Brown, J. S., Burton, R. R., & deKleer, J. (1982). Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II, and III. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems.* New York: Academic Press.

Bunderson, C. V. (1974). The design and production of learner controlled courseware for the

TICCIT system: A progress report. *Journal of Man-Machine Studies, 6,* 479–491.

Burton, R. R., & Brown J. S. (1979). An investigation of computer coaching for informal learning activities. *International Journal of Man-Machine Studies,* 5–24.

Collins, A. (1977). Process in acquiring knowledge. In R. C. Anderson, R. J. Spiro & W. F. Montague (Eds.), *Schooling and acquisition of knowledge*. Hillside, NJ: Lawrence Erlbaum Associates.

Dennis, J. R., & Kansky, R. J. (1984). *Instructional computing*. Glenview, IL: Scott, Foreman & Co.

Gagne, R. M., Wager, W. & Rojas, A. (1981). Planning and authoring computer assisted instructional lessons. *Educational Technology, 21,* 17–26.

Grimm, P. (1983). Attention, perception and memory: Screen design for CBT. *Data Training,* 41–42.

Heines, J. M. (1983). *Screen design strategies for computer aided instruction*. Bedford, MA: Digital Press.

Jenkin, J. M. (1981). Some principles of screen design and software for their support. P. R. Smith (Ed.), *Computer Assisted learning*. Oxford, England: Pergamon Press.

Kearsley, G. (1984). Authoring tools: An Introduction. *Journal of Computer Based Instruction, 11(3),* p. 67.

Kearsley, G. (1982). Authoring systems in computer based education. *Communication of the ACM,* 25(7), 429–437.

Kearsley, G. (1984). Microcomputer software: Design and development principles. *Journal of Educational Computing Research, 1* (2), 215–226.

Merrill, M. D. & Wood, L. E. (1984). Computer guided instructional design. *Journal of Computer Based Instruction, 11* (2), 60–63.

O'Neal, A. F., O'Neal, H. L. (1979). Author management systems. In H. F. O'Neal (Ed.), *Issues in instructional systems design*. New York: Academic Press.

Sleeman, D., & Brown, J. S. (1982). *Intelligent tutoring systems*. New York: Academic Press.

Steinberg, E. R. (1984). *Teaching computers to teach*. Hillsdale, NJ: Erlbaum.

Stevens, A., & Roberts, B. (1983). Quantitative and qualitative simulation in computer based training. *Journal of Computer Based Instruction, 10* (1&2), 16–19.

Walker, D. F., & Hess, R. D. (1984). *Instructional software*. Belmont, CA: Wadsworth.