

INCREASING CBT DEVELOPER PRODUCTIVITY WITH AN INSTRUCTIONAL DATABASE ¹

Jesse M. Heines, Ed.D.

Asst. Prof. of Computer Science
University of Lowell
Dept. of Computer Science
One University Avenue
Lowell, MA 01854

Larry Israelite, Ph.D.

Director, Training Products
Scientific Systems, Inc.
One Alewife Place
35 CambridgePark Drive
Cambridge, MA 02140

Abstract

Computer-based training (CBT) course developer productivity is most often expressed as the number of development hours needed to produce one hour of on-line instruction. CBT authoring systems offer a number of approaches to increasing developer productivity, generally by providing built-in features for typical instructional sequences. As one would expect, each approach has its benefits and shortcomings. We have adopted an "instructional database" approach, which attempts to provide simplicity for the course developer without sacrificing true programming power. In addition, this approach is designed to take advantage of the desirable built-in features of the authoring language while maintaining the extensibility needed to implement new instructional designs.

¹ This paper was the recipient of the Best Paper Award at the 28th International Conference of the Association for the Development of Computer-based Instructional Systems (ADCIS) in November 1986.

Measuring CBT Developer Productivity

The major purpose of computer-based training (CBT) authoring systems is to increase CBT developer productivity. One standard measure of this productivity is the number of development hours needed to produce one hour of on-line instruction. Measures quoted in the literature typically fall between 50 and 150 hours of development for each hour of on-line instruction, but can range to as many as 300 hours for complex equipment simulations.

It is often difficult to reconcile these measures with claims made by authoring system vendors, who typically dramatically underestimate the time taken to produce highly creative, *finished* instructional products. When you hear claims that a certain authoring system allows developers to produce 50 screens per day, you can be assured that that includes neither adequate design or testing time nor the realities of the production environment, i.e., insufficient access to subject matter experts, equipment down time, etc. It is our experience that even the best CBT developers can produce only one or two instructional *sequences* per day, where a sequence is defined as a series of 6-10 related displays, each requiring a non-trivial response (i.e., requires answer analysis) and each evolving from the previous display without a full screen erase.

Productivity and Authoring Systems

Regardless of what you include when you tally up your development hours, the key to increasing developer productivity is to reduce the number of hours needed to produce each hour of on-line instruction. The most common means used in pursuit of this end is the adoption of a CBT authoring system. Such systems help enhance productivity in a number of ways. The list that follows highlights the most typical approaches. We have included comments on the benefits and shortcomings of each.

1. **A language-less system for specifying typical screen displays and their appropriate presentation sequence.**

The main shortcoming of this approach is that it addresses only the most visible aspect of CBT: what students see on their screens. It severely limits the developer's ability to address the more significant aspect of on-line instruction: the implementation of meaningful student/ computer interactions.

Some systems that employ this approach contain very nice graphics editors with special routines for screen wipes, animation, etc. In general, however, we have only found these systems to be useful in creating the most elementary of instructional sequences, e.g., those that demonstrate a process by displaying a linear sequence of "slides" and asking the student to press RETURN to go from one to the next.

Language-less systems are typically tightly "bounded," i.e., they do not allow programming outside of the features they supply themselves. As such, we have found these systems too restrictive for all but the simplest of interactions. Most of these systems are geared toward beginning developers, but some are now appearing with enhanced functionality for more sophisticated applications.

2. **A simplified language for specifying typical instructional sequences.**

Simplified language systems were devised to reduce the programming skill needed to produce CBT courseware. Such languages typically lack one or more of the basic components of a computer language, e.g., the ability to declare variables, to create looping and if/then structures, or to segment a program via subroutines. Like language-less systems, these systems can be useful for beginners, but we have found that developers quickly grow out of them as they gain experience. Rather than trying to make developers into programmers, we recommend that organizations employ a team approach to courseware development, with experienced programmers assigned to support instructional and subject matter experts who are designing CBT materials (Heines, 1985; Heines & Moreau, 1980).

The benefit of simplified language systems is that most of them employ a very clean and easy-to-learn syntax for typical CBT constructs such as multiple choice questions. Their shortcoming, however, is that (like language-less systems) they are usually tightly bounded and severely restrict creative extensions. In other words, they make the simple trivial and the difficult impossible. We have found that it is usually easy to write our own templates or subroutines in sophisticated authoring systems to implement typical CBT constructs, thus gaining the main benefit of using a simplified language system without begin limited by its non-extensibility.

3. **A set of program templates for implementing typical instructional sequences.**

A template is a program shell that is fully operational but lacks subject matter content. Templates are typically devised for authoring systems that do not have subroutine capabilities. They allow developers to implement their designs within the constraints of a predefined instructional sequence, simplifying the programming task.

The principal advantage of using templates over simplified languages is extensibility. If a developer requires a new type of instructional sequence, a programmer can usually implement a template to achieve the desired effect within a matter of hours. Course developers then need only insert their subject matter content into the the program code in the appropriate places to use the template in their courseware. Once a template is developed, it can be used any time an instructional design calls for its approach by plugging in the appropriate subject matter.

The principal disadvantages of templates are that they foster large programs by requiring code to be repeated each time it is needed and, consequently, make revision of such programs difficult because a change to a template requires changes in each section of code in which that template is used.

4. **A set of utility programs and subroutines for implementing typical features.**

Alfred Bork and his colleagues at the University of California at Irvine write most of their CBT materials in Pascal. Bork has stated that they use Pascal "not because it's a good CBT authoring language, but simply because it's a good language" (Bork, 1981). If you look at Bork's code, however, you will see that it consists mostly of calls to subroutines that implement screen management, testing strategies, and many of his standard instructional sequences. Bork and his colleagues have built up a sizable library of utility programs and subroutines that they use as building blocks when creating their courseware.

This approach is, in fact, typical practice for virtually all experienced software engineers. One seldom writes a program completely from scratch, and most software specialists draw on a personal as well as public library of common routines to use as program building blocks. This approach provides both the simplicity and extensibility of program templates, but usually results in a smaller final program than the template approach. (The template approach requires a complete copy in each instructional sequence that uses it, while only one copy of a subroutine is needed regardless of the number of times it is used.)

The shortcoming of this approach is that most CBT authoring languages don't provide full subroutining capabilities with parameter passing and local variables. One therefore has to move to a general purpose language like Pascal and re-invent all of the features inherent in even simplified CBT authoring languages, such as response analysis, student record keeping, etc. More sophisticated features, such as author-defined variable pitch character sets, may even require the writing of assembly language routines, a tedious task even for experienced programmers. In addition, even relative simple code modifications can require the help of experienced programmers. Thus you might gain the benefit of good software engineering but lose the benefit of the efforts that have gone into developing the useful features inherent in CBT authoring systems.

The major trade-offs one must consider in choosing an authoring system are therefore simplicity vs. power and built-in features vs. extensibility. The next section describes a somewhat different approach with which we have been experimenting, the construction of an "instructional database." This approach attempts to minimize trade-offs and attempts to provide the best features of each of the four approaches described above.

An Instructional Database Approach

Scientific Systems, Inc., (SSI) is often asked to develop courseware employing standardized instructional sequences. One of the most common of these sequences is one in which students are asked to name the parts of a complex machine, locate them in a diagram or photograph, and state their functions. SSI has developed a standard approach to "part definition" task and has implemented its approach on both TICCIT and the IBM PC.

In the past, SSI course developers have researched the subject matter, specified the names, locations, and functions of the machine parts to be taught, and provided this information to programmers for course implementation. Since the developers write their specifications on-line, SSI wished to devise a method for automating the specification-to-program step and eliminate redundant data entry and programming. Our solution was to design an instructional database. Using this approach, developers enter their course specifications in a simple format using any standard word processor. Their raw specification files are then read by a program that converts them into a form which can be read directly by a running CBT program. The program that reads the raw specification files functions as a "front-end" to the CBT program, allowing the CBT program to access any part definition record in the instructional database with a surprisingly small subroutine.

The beauty of the instructional database approach is that it provides both simplicity and power. Course developers see the simplicity of the system by working with it at the

database creation level, which can be accomplished with any standard text editor. They can use the front-end program to check their raw specification files for syntax errors and see how their displays will look when processed by the CBT program. If they are not satisfied with the results, they can make corrections and/or alterations in their original files by re-editing their files with the text editor of their choice.

Programmers see the power of the system by working with it at the database reading level, after it has been preprocessed by the front-end program. The information in the database can be used in a large number of ways. For example, the same information can be used to generate:

- linear slide show-like sequences for introductions and reviews,
- help sequences that explain the location and function of a named machine part,
- practice exercises that require students to indicate the location of a part given its name or function,
- practice exercises that require students to supply the name of a part given its location or function, and
- test items that match stated functions with part names and/or locations.

The instructional database approach also provides a wealth of built-in features with complete extensibility. The front-end program is written in Pascal, and we have implemented versions that run under both VMS on VAX systems and DOS on IBM PCs. The database reading program is written in TenCORE, a powerful PLATO-like authoring system for the IBM PC available from Computer Teaching Corporation in Urbana, Illinois. We can therefore take advantage of all of the authoring system's built-in features because the courseware is ultimately presented using by that system. In addition, we can add our own features (like automatic text centering and generation of lists of bullets) by modifying the front-end. We can also create new instructional sequences by writing new subroutines to use the information in the instructional database in novel ways.

Productivity and the Instructional Database

The instructional database approach has completely eliminated the need for specifications written by course developers to be retyped by programmers. Once the specifications are run through the front-end program, at least one aspect of the course is completely ready for presentation without further programming. Our programmers thus concentrate on providing a library of standard routines rather than implementing each instructional sequence designed by the developers. When developers do specify a new design, that routine becomes part of the library and can be used for any subject matter.

Thus we believe that we have achieved the benefits of both worlds by employing an approach that is easy for non-programmers to use but does not inhibit their creativity. At the same time, the approach is very efficient from a programming point of view and readily extensible to accommodate new designs. We are now working on building our library of

standard routines and assessing the full impact of this approach on the number of hours needed to produce each hour of on-line instruction. If the results of these efforts are as positive as we anticipate, we hope to achieve very significant increases in CBT developer productivity for a wide range of instructional applications.

References Cited

Bork, Alfred, 1981. *Learning With Computers*. Digital Press.

Heines, Jesse M., 1985. Anybody Can't Do CBT. *Training News* 6(7):9.

Heines, Jesse M., and Ken Moreau, 1980. The Three-Pronged Computer-Based Course Development Process. ADCIS Proceedings, Washington, D.C.