Automated Instructional Development Using Personal Computers: Research Issues

Greg Kearsley
Courseware Incorporated

Abstract. This article discusses the design of systems for automating instructional development. A prototype automated Instructional Systems Design (ISD) system called the *Courseware® Instructional Toolkit™* is described. The purpose of this system was to explore the extent to which ISD could be automated and the research issues associated with such automation. The major research issues identified were: (a) what aspects of ISD can be automated, (b) transfer of information between ISD tasks, (c) effects on ISD processes and outcomes and (d) implications of automated systems for the skills of instructional developers.

Introduction

Instructional development is a very labor-intensive process. Because of this, it is very expensive, susceptible to individual differences in skill levels, and not especially reliable. Despite the emergence and widespread use of Instruction Systems Design (ISD) procedures (Logan, 1981; O'Neil, 1979), instructional development is still not a highly refined activity. There is a recognized need to improve the quality of ISD methods (Montague, Wulfeck, & Ellis, 1983).

One potential solution to the problem is to automate instructional development procedures. Over the past decade, a number of research projects have examined this possibility. Braby and Kincaid (1981) developed a computer-based editing system that helped authors create Navy technical manuals. Brecke and Blaiwes (1981) described the CASDAT system that was intended to automate instructional analysis and design activities for technical training. Kincaid, Braby & Wulfeck (1983) discuss computer aids for testing. O'Neal and O'Neal (1979) describe the Author Management System developed to automate project management in large-scale ISD projects. Merrill and Wood (1984) describe their Lesson Design System implemented on an Apple II microcomputer.

There are a number of different facets to automation and instruction (Kearsley & Seidel, 1985). A great deal of attention has been given to automating one particular aspect of instructional development, namely, the creation of computer-based instruction programs using authoring systems. Authoring systems allow computer-based programs to be developed without the need to know a computer programming language, and reduce the development time required (Kearsley 1982; 1984). There are literally hundreds of authoring systems available for personal computers, each one with different features and capabilities. Many instructional developers now use these systems to create computer-based instruction. However, authoring systems represent only the tip of the iceberg, even in the development of computer-based instruction. More time is required for the analysis, design, and evaluation of instruction than the actual creation of programs. To make more significant gains in reducing the time (and expense) associated with instructional development, it is necessary to automate these other tasks. While many existing authoring systems could serve as a good basis for more comprehensive automated development systems, at present they are limited to producing programs.

Another important aspect to automating instruction is the production of print and multimedia materials. Personal computers are now commonly used to create, edit and typeset textbooks and manuals as well as to generate graphics, slides, overhead transparencies, and animated sequences. As instructional developers become increasingly accustomed to using computers for media production tasks, it will become natural to use the same systems for analysis, design, and evaluation tasks.

To summarize, the instructional development field is being driven towards the use of computers for all phases of ISD by a number of factors including the desire for better quality results, the need for increased efficiency, and for convenience. Like other domains prior to the use of technology, there is little understanding in the ISD field of what kind of impact automation could have. The purpose of this article is to identify research issues in the development of automated ISD systems based on a prototype system called the *Courseware® Instructional Toolkit™*.

Architecture of the System

For the purpose of discussing the research issues involved in automating ISD, it is not necessary to extensively describe the Toolkit prototype. The prototype served primarily as a vehicle to identify and study the issues. It also served as a means to investigate the feasibility and resources required to develop a full scale system. On the other hand, the particular nature of the system undoubtedly influenced the kinds of issues raised. For this reason, the basic structure and characteristics of the system will be briefly described.

It should be emphasized that the Toolkit is intended for use in large scale ISD projects that involve a team of developers working together to produce a substantial amount of curriculum material (i.e., hundreds of lessons) under time pressure. In this environment, standardization and communication of results from one task to another is very important. While the Toolkit could be used by a solitary developer for a single course, most of the functions would probably not be worthwhile since the tasks could be completed faster and more easily by manual methods.

The Toolkit was implemented on an IBM PC/XT and consisted of four different types of software (see Figure 1).

The system manager integrated all of the other programs in the Toolkit and allowed them to exchange data. It also allowed us to store keystroke or command sequences (needed for macro capabilities—described below) and provided "windowing" capabilities across all programs. We used DESQ from Quarterdeck Office Systems, although there are a number of other similar programs now available for the IBM-PC.

Applications software includes commercially available programs for word processing, database management, spreadsheets, project management, statistical analysis, telecommunications, etc. Authoring software includes any commerically available authoring language or system, as well as programs for graphics creation, typesetting, and slide production. Custom designed software refers to programs that had to be written in a programming language to accomodate unique requirements that could not be met by the commercially available applications or authoring programs. Figure 2 shows the specific ISD functions that we explored in the prototype. Functions with an asterisk were actually implemented; those without were designed only on paper. These particular functions were selected for one of three reasons: (a) they represent commonly performed ISD tasks, (b) they represent ISD tasks typically performed poorly, or (c) they represent ISD tasks that are seldom performed (but should be).

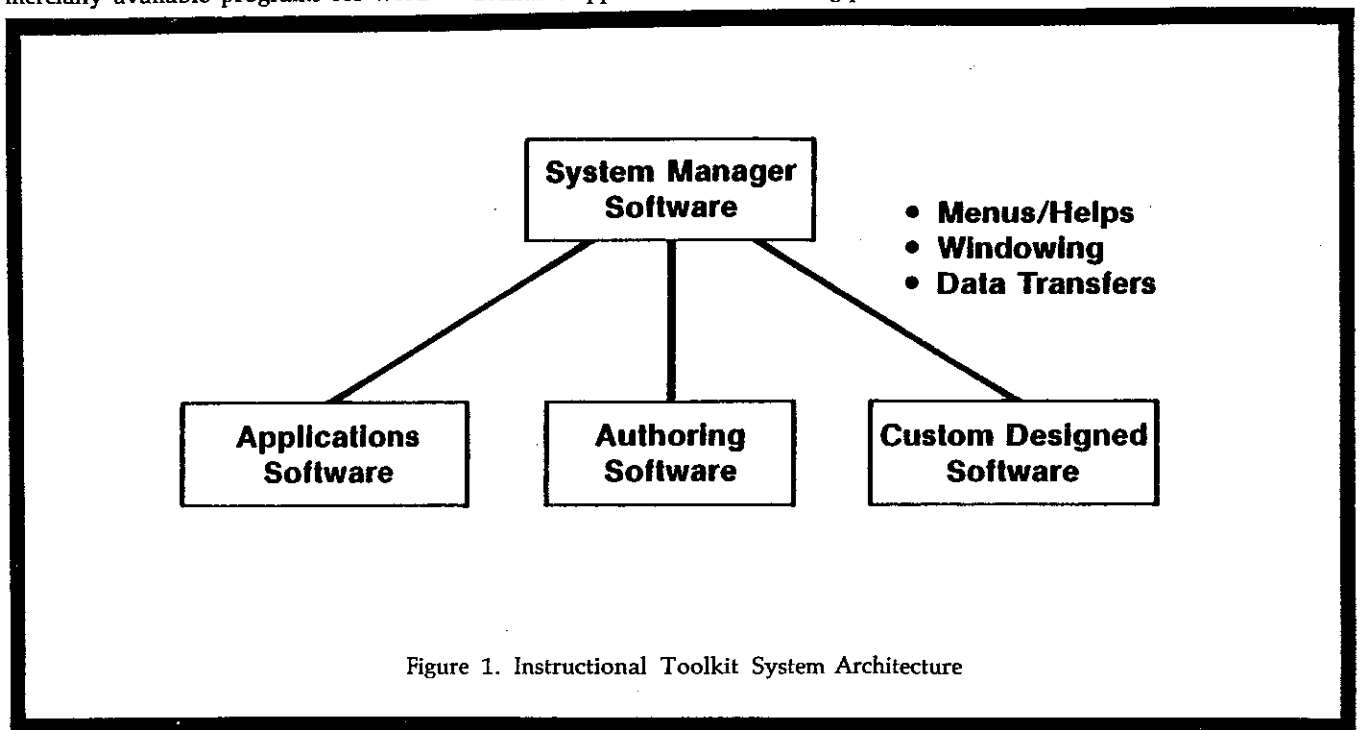

Figure 1. Instructional Toolkit System Architecture

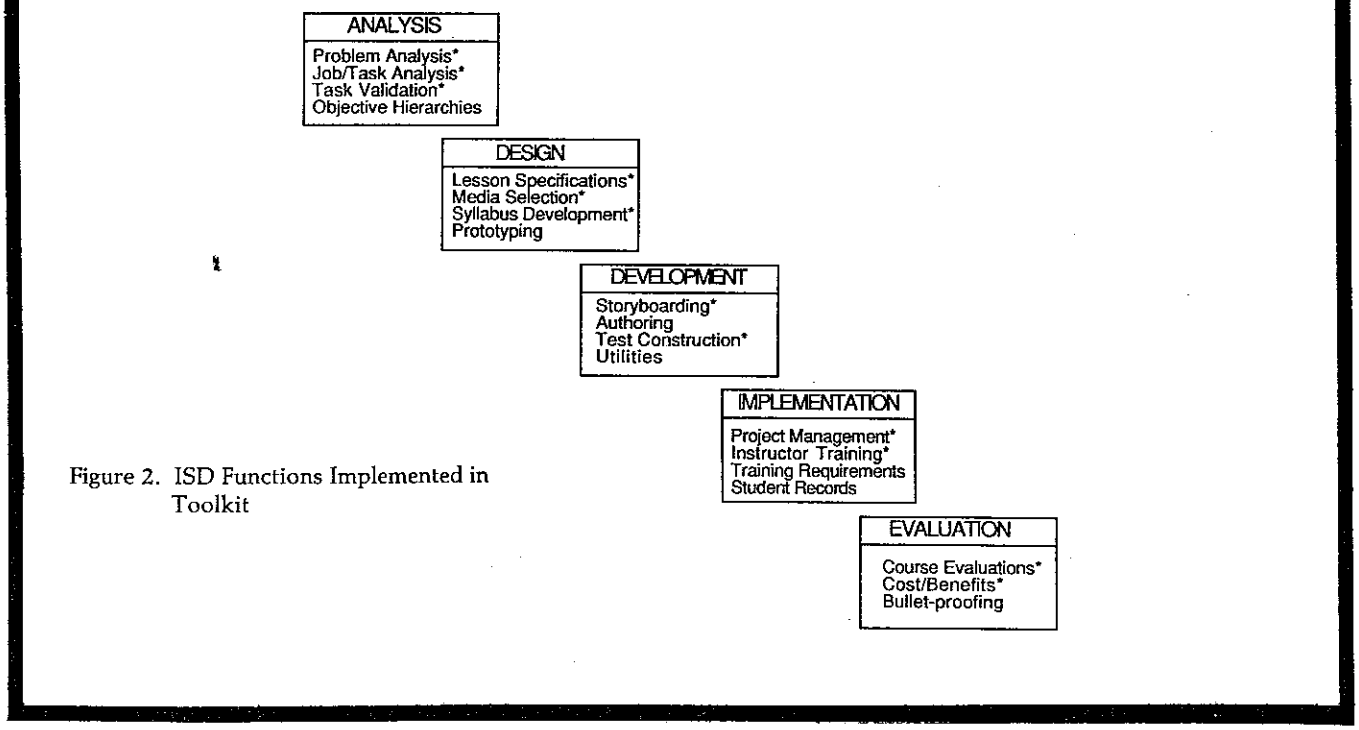

Figure 2. ISD Functions Implemented in Toolkit

Associated with each ISD function is a standard set of components: overviews, macros, shells, examples, job aids, and helps. The overview explains the nature and rationale of the ISD task and relates it to other ISD tasks. Although the overviews in the prototype were for reference only, they could be elaborated into full interactive tutorials that teach the ISD task to new developers. Macros represent the automated procedures and consist of stored command sequences appropriate for the applications or authoring program being used for that ISD function.

Shells represent an empty template or outline that can be completed (on-line or off-line) to accomplish the particular ISD task involved. For example, the shell for the problem analysis function, is the outline of a problem analysis document to be filled in using a word processing program. The shell for the task hierarchy function is a blank set of boxes to be filled in using a graphics program. The shell for a cost/benefits function is an empty spreadsheet template.

Examples are shells that have been completed for specific instructional projects. They serve as models or illustrations of what the task looks like when completed. Job aids are descriptions of the steps or procedures to be followed to complete an ISD task. In some cases, the job aid is simply a shell with a listing of the questions to be answered to complete the task. Helps explain how to use the current function. This could be an explanation of what the macros do or a specific step in the current task.

Examples of Use

A few examples will help clarify how a developer uses the system. Figure 3 shows a schematic of how lesson specifications are written. The lesson specification function in the Toolkit prototype used a word processing program as its basis. After selecting the lesson specification function, the developer can choose to see an overview, example, shell or job aid. There are three ways the designer could complete the task: (a) select a shell and fill in the appropriate information under the headings provided, (b) select an example and use this as a model to complete the shell, or (c) select the job aid and answer the questions to fill in the shell. Because each component can have its own window, it is possible to keep the example or job aid on one part of the screen while filling in the shell.

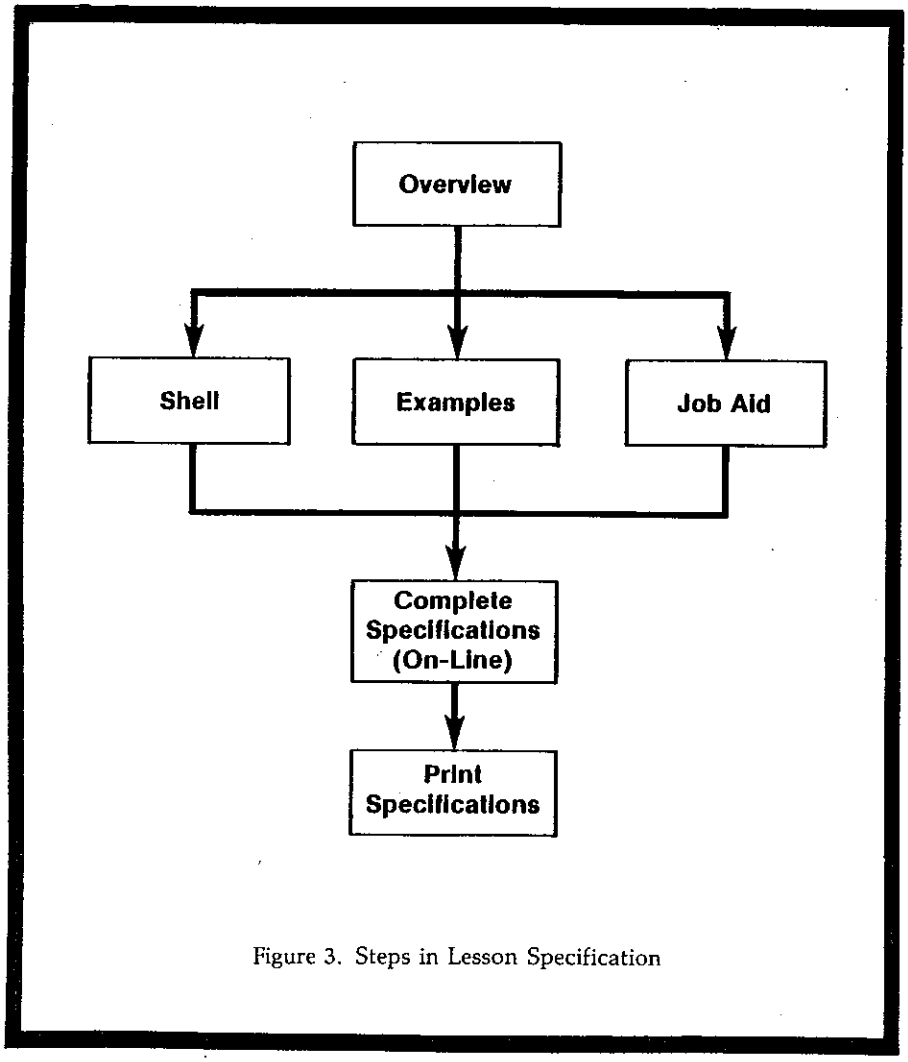

Figure 3. Steps in Lesson Specification

Once the specification has been completed, it can be printed out for review. If the developer is working with others on a network, the specification could be viewed on the screen. Since communications can be brought up in its own window at the same time a specification is being reviewed, it would be possible for two developers to have an on-line conversation about the specification. Because we did not have a net-worked system, we were unable to explore this interesting possibility.

Many of the analysis, development, and evaluation functions use word processing programs and work in a fashion similiar to the specification function (e.g., problem analysis, job/task analysis, instructor training, test construction). Other functions, such as task validation, syllabus development, media selection, course evaluations, and student records use database management programs.

Figure 4 shows the sequence of steps in a syllabus development task. To construct a syllabus, the developer begins by transferring objectives to the database shell from an objectives hierarchy or task listing already created. The shell has columns corresponding to the critical parameters that will be used to organize the objectives into units or lessons (e.g., prerequisites, type of learning, duration). The developer must specify the parameters and a rule for combining objectives based upon the parameters (e.g., hands-on objectives at end of unit, maximum of 20 minutes per unit).

Once these parameters have been indicated for each objective, the program sorts the objectives into units based upon the rule provided. The result is then displayed to the developer who can change the rule or parameters and rerun the program. When the designer is satisfied with the syllabus, it can be printed out.

All of the operations (specifying parameters and rules, sorting, and printing) are handled by macros. The syllabus development function also includes the other standard Toolkit com-

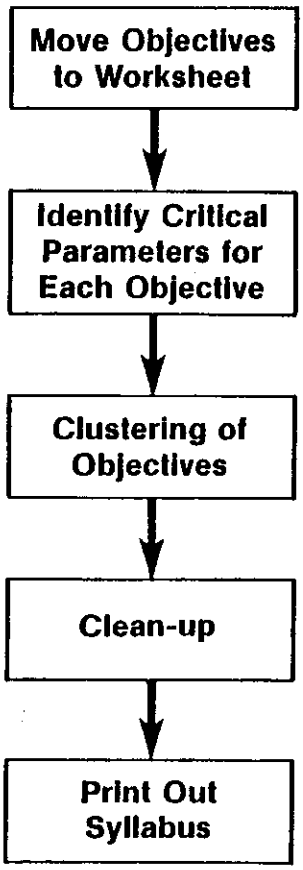

Figure 4. Steps in Syllabus Development

ponents (overview, examples, job aid, and helps). Using this function, a large syllabus encompassing thousands of objectives and hundreds of units can be created in a matter of hours compared to days by manual methods. Furthermore, the process is repeatable—given the same parameter values and rules, the program produces the same syllabus each time.

The development of questionnaires for course evaluations is another function that uses a database management program. To compose a questionnaire, the developer simply selects the items to be included. The items can be open-ended questions as well as multiple choice or ratings. New items can be added by bringing up a word processing program in another window. After the questionnaires are completed and the responses entered, a statistical analysis program could be used to analyze the results and keep these in the course evaluation database.

Project costing (a project management task) is an example of a function that uses a spreadsheet program. Other functions that use spreadsheet programs as their basis are training requirements estimation and cost/benefits analysis. The shell for this program is a template that requires the developer to fill in values for the estimated number of objectives or course duration, type of media involved, complexity of content, distance between customer and project staff, and other critical project costing parameters. Once these values have been filled in, the program uses a matrix of historical data to compute the estimated number of person-days required for different job categories and costs based upon salary and overhead rates.

The developer can model different assumptions about course size, media, and project staffing to produce a range of cost estimates. Using this function, project costing can take a few minutes to complete instead of hours using a calculator. More importantly, because the function uses cumulative historical data, the estimates are likely to be more accurate than ones made on the basis of a single individual's experience.

Cost/benefits analysis is another example of a function that uses a spreadsheet program. The developer first completes the system and development costs and then estimates the utilization of the program. The program then calculates the cost per student hour. If any of the entries are changed, the calculation is immediately redone. This allows the developer to explore different assumptions about costs or utilization.

Issues

Putting aside the obvious practical benefits alluded to in the previous section (i.e., standardization, time savings, increased reliability, more accuracy), the Toolkit prototype raises some interesting research questions about

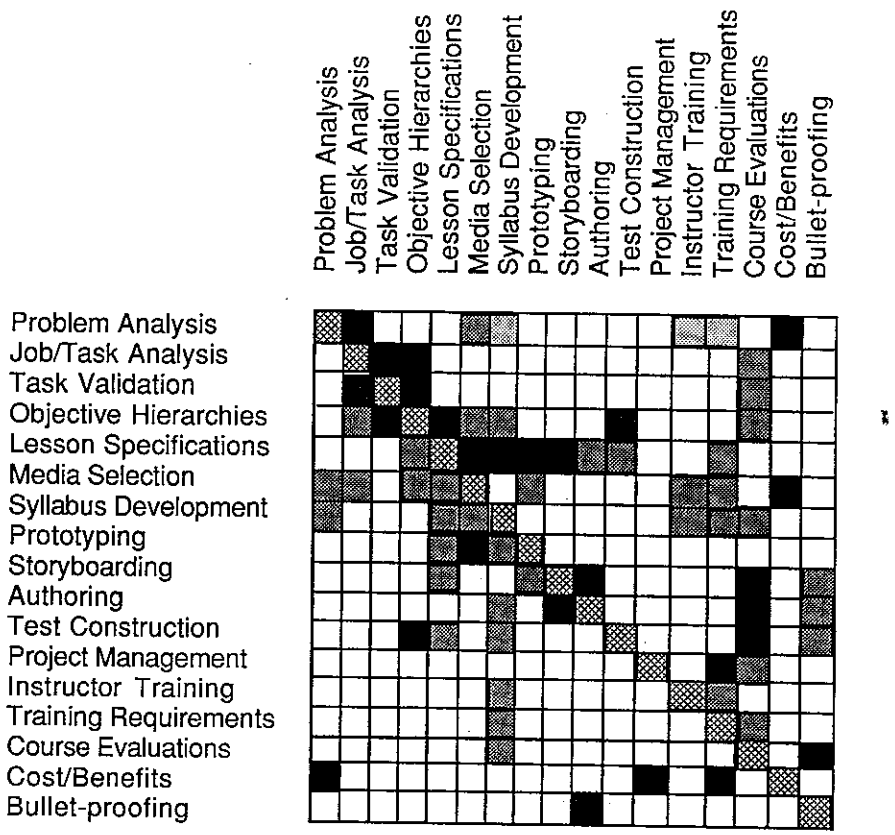

Figure 5. Predicted Relationships Between ISD Tasks

automated instructional development. They are as follows:

1. What aspects of ISD can/cannot be automated?

This is a classic issue in the field of human factors engineering that deals routinely with performance limitations in human-machine systems. It is also an important question in the field of artificial intelligence, which deals with the design of computer programs capable of "intelligent" actions. Both of these perspectives are relevant to the issue of what aspects of ISD can/cannot be automated.

Most ISD tasks can be broken down into a relatively simple clerical component and a more complex judgemental component. For example, in lesson specification, certain questions must be asked and the answers obtained (e.g., "What is content for this objective?"; "How could you illustrate this concept visually?", etc.). This is a simple clerical task. The answers must then be translated into a specification for a definition, example, test question, review, and so on. This step requires judgement. In syllabus development, the judgement comes in identifying the parameters of each objective and the grouping rules; the sorting into units is clerical in nature. In project costing, the judgements are made in estimating the size and complexity of the project; the generation of the actual cost estimates is mere calculation.

The clerical components of a task are relatively easy to automate. They consist of procedures with a fixed set of steps, formulas, or decision rules. On the other hand, the judgemental components are more difficult to automate. They depend upon context-dependent rules, very detailed discriminations, or heuristic strategies. They are often difficult to state explicitly.

The clerical component is very susceptible to error. A general finding of human factors research is that most people produce high failure rates in repetitious tasks due to fatigue, boredom, and loss of attention. For this reason, automation of the clerical component is usually a good idea.

Automation of the judgemental component depends heavily on artificial intelligence approaches (including expert systems). For example, consider the task of making decisions about the successful use of computer-based training. Judgements about the suitability of a course for computers or the appropriate kind of system to use depend upon a series of inter-related rules derived from actual computer-based training projects. These rules can be put in the form of an expert system that provides training managers with advice for specific instructional applications (Kearsley, 1985).

The Toolkit prototype focused on the clerical component of ISD tasks. More sophisticated software and a much more detailed understanding of the judgemental process in instructional development would be needed to go beyond this level.

2. What are the relationships among different ISD tasks?

The Toolkit was designed as an integrated system that allowed the results of one ISD task to be passed forward or backward in time to another ISD task. This raises the issue of the usual or possible data transfer paths among different ISD tasks. When ISD is done manually, the flow from one task to another is

seldom identified. Furthermore, since it typically takes place over a period of days or weeks, it would be difficult to track. In an automated ISD system, however, the transfer of data between tasks can occur in a matter of seconds. Since we would like to automate as many transfers from task to task as possible, these relationships are of considerable importance in an automated system.

Figure 5 is a theoretical plot of the frequency of transfers between the 17 ISD tasks in the Toolkit prototype. Each cell indicates the likelihood of an interaction between one task and another. The darker the cell at the intersection of any two tasks, the greater the degree of interaction predicted between them. For example, some transfers are routine (e.g., moving objectives to syllabus development or test construction, lesson specifications to storyboarding, evaluation data to lesson specifications) while others are rare. Note that the plot is asymetrical; transfers have directionality. Most interactions are predicted to occur in a forward direction (the top left diagonal of the matrix) rather than backward (bottom right diagonal). To validate this plot empirically, it would be necessary to count the frequency of task interactions in the Toolkit across time

Another important part of this data transfer issue is whether there are "optimal" ISD paths for a given instructional project and goals (e.g., least time/cost, best quality outcomes, smallest project team). By conceptualizing ISD tasks as nodes in a network connected by their transfer paths, it is possible to examine optimal paths using various network analysis methods. The network will usually be different for each project because projects include different subsets of ISD tasks and start/stop at different tasks.

3. How will automation affect the process and outcomes of ISD?

There are many ways that automation could change how ISD is conducted and the results of the process. For example, the availability of ISD functions in an automated system could encourage their use. ISD tasks that are tedious or difficult to do manually are likely to be skipped. In an automated system, such tasks may be carried out because they are less tedious or difficult. Thus, one of the possible effects of automated ISD systems is that more ISD tasks would be completed for a given project.

In an open architecture system like the Toolkit, it is possible that a developer might have a choice of two or more functions for a given ISD task. For example, in the prototype, there were two levels of media selection: a macro analysis for making judgements based upon the attributes of an entire course, and a micro analysis based upon the attributes for every objective of a course. In this situation, the developer must select which function to use (or try both). This requires the developer to make a judgement about the relative merits of one ISD method over another for a given project. Will developers make the best choice, and if they make inappropriate choices, how and when can this be detected?

This raises a fundamental point—how can we measure the quality of ISD outcomes in order to detect the effects of automation? The ultimate test of ISD quality is the extent to which the instruction works, i.e., helps people learn. It would seem a simple matter to assess the effectiveness of ISD methods by measuring differences in job performance or achievement. Of course, there are too many intervening variables to make this possible. A more feasible test for automated ISD is to compare the results of a particular task done manually and via the system. In other words, if one developer completes the task manually and another developer (of matched experience) completes the task using the system, what are the differences in the resulting outcomes? Do they differ in completeness, or correctness? Is a syllabus or lesson produced by a developer using the Toolkit as good, better or worse than one produced by the same developer without the system?

From our work with the Toolkit prototype, it is clear that one of the major effects of an automated ISD system is on the time required to complete ISD tasks. The use of the Toolkit produced time savings in the order of 50 percent for some tasks. Furthermore, the delay between finishing one task and beginning another can be minimal. On the other hand, some tasks took longer because developers experimented with different alternatives or variations. In addition, when there were problems with the system or the hardware, no work was accomplished. It is apparent that the use of automated ISD systems will change the timescale of instructional development, although not necessarily reduce the time required.

4. What are the implications of automated ISD systems for the skill levels of instructional developers?

One enduring misconception about technology is that its major purpose is to substitute for human abilities and replace people. In fact, the rationale for using technology is always to improve productivity by augmenting or extending human abilities. In the case of automated ISD, we are interested in creating tools that allow developers to accomplish more efficient or effective instructional development. The system should compensate for the lack of experience of a new developer. The system should also permit experienced developers to leverage their skills and eliminate time wasted in routine aspects of ISD tasks.

However, the effective use of an automated system requires mastery of the system. In the case of the Toolkit prototype, it was necessary for developers to attain a general level of computer literacy as well as obtain some familiarity with the specific application programs used (i.e., word processing, database management, spreadsheets, etc.). A considerable amount of time was required to learn how to use all of the features and capabilities of the system. For developers to take advantage of the system and use it productively, this time

Automated ISD prototypes like the Toolkit help us explore new ISD capabilities that are not possible without the computer.

investment was essential.

One of the new skills needed by developers is the ability to select one approach versus another for a given training project. Most traditional training of instructional developers tends to focus on the use of a particular ISD methodology. However, in an automated ISD system such as the Toolkit, it may be necesssary to choose the most appropriate ISD function from a selection of possibilities. This suggests that developers will need a more comprehensive understanding of ISD in order to use an automated system effectively.

The implication of these observations is that an automated ISD system may be more valuable to an experienced developer than a novice. This consititutes a hypothesis that could be tested in further research.

Conclusions

Because of time, budget, and equipment limitations, it has not yet been possible to extensively field test the Toolkit prototype in actual ISD projects. Various parts of the Toolkit have been used in projects and have led to the observations and hypotheses described in this article. The prototype has demonstrated that an automated ISD system provides a rich environment for conducting research on instructional development methods.

One intriguing aspect of automating ISD that was not explored in the prototype is the development of functions for tasks that cannot be done well by manual means. For example, Figure 2 lists functions for prototype development and bullet-proofing. The prototype development task involves creating a sample of a lesson or course before development starts. This task is important to avoid misunderstandings about the format or organization of materials. However, because making a prototype is time consuming and expensive, this step is often skipped (with dire consequences). However, there are programs available that make it possible to easily create an electronic "mock-up" of a printed, audiovisual, video or computer lesson that could be used as a prototype.

Bullet-proofing involves testing for all the content and logical flaws in instructional materials, especially interactive programs. To test a lesson for all flaws, it would be necessary to generate every possible student response. It is easy to write a computer program than will generate every possible response at random. It is also possible to write a program that simulates a student and makes likely responses for each instructional sequence based upon typical learner behavior (including misconceptions and errors). Traditional bullet-proofing technique depends upon field testing that is very tedious and time-consuming. Furthermore, traditional bullet-proofing methods are not very comprehensive and identify only a small fraction of problems.

The point of these two examples is that automated ISD could help us explore new ISD capabilities that are not really possible without the computer. The use of computers has already led to the development of new theory in many areas of instruction (Kearlsey, Hunter & Seidel, 1983). There is no reason why it couldn't also have this effect in the instructional development area.

The major stumbling block in the development of automated ISD systems is our lack of detailed understanding of the instructional development process. In order to write a program to carry out an ISD procedure, it must be possible to describe it explicitly. However, much ISD methodology is based upon intuitive judgements and tacit knowledge. In order to automate such processes, this intuition and tacit knowledge must be made explicit. One of the valuable outcomes of research in automating ISD is likely to be a more complete understanding of the creative steps in instructional development. It seems ironic that in creating programs to mimic human behavior, we should learn more about it.

One of the valuable outcomes of research in automating ISD is likely to be a more complete understanding of the creative steps in instructional development.

Acknowledgement: The Instructional Toolkit prototype was the joint effort of many employees of Courseware Inc. In particular, Nile Gardner, Diana Pickens, Susan Lindgren, and Kathy Rose made significant contributions to its design and development.

References

Braby, R., & Kincaid, J.P. (1981). Computer aided authoring and editing. *Journal of Educational Technology Systems, 10*(2), 109-124.

Brecke, F., & Blaiwes, A. (1981). CASDAT: An innovative approach to more efficient ISD. *Journal of Educational Technology Systems, 10*(3), 271-283.

Kearsley, G. (1985). The CBT advisor: An expert system program for making decisions about CBT. *Performance & Instruction, 24*(9), 15-17.

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

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

Kearsley, G., Hunter, B., & Seidel, R.J. (1983). Two decades of CBI research: What have we learned? *T.H.E Journal, 11*(6 & 7).

Kearsley, G., & Seidel, R.J. (1985). Automation in training and education. *Human Factors, 27*(1), 61-74.

Kincaid, J.P., Braby, R.S., & Wulfeck, W.H. (1983). Computer aids for editing test questions. *Educational Technology, 23*, 29-33.

Logan, R.S. (1981). *Instructional Systems Design*. New York: Academic Press.

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

Montague, W.E., Wulfeck, W.H., & Ellis, J.A. (1983). Quality CBI depends upon quality instructional design and quality implementation. *Journal of Computer-Based Instruction, 10*(3&4), 90-93.

O'Neal, A.F., & O'Neal, H.L. (1979). Author management systems. In H.F. O'Neil (Ed.), *Issues in Instructional Systems Design*. New York: Academic Press.

O'Neil, H.F. (1979) *Procedures for Instructional Systems Design*. New York: Academic Press.