AUTOMATING INSTRUCTIONAL DESIGN:
APPROACHES AND LIMITATIONS

J. Michael Spector and Celestia Ohrazda
Syracuse University

26.1 INTRODUCTION

In the last half of the previous century, many tasks that had been regarded as best accomplished by skilled workers have been shifted partially or entirely to computers. Examples can be found in nearly every domain, including assembly line operations, quality control, and financial planning. As technologies and knowledge have advanced, the tasks of scientists, engineers, and managers have become considerably more complex. Not surprisingly, there has been a tendency to apply computer technologies to the more complex and challenging tasks encountered by the user. Instructional design (ID) represents a collection of complex and challenging tasks.

This discussion reviews the history of automation in the domain of ID. An overview of automation in the domain of software engineering is provided, which introduces key distinctions and types of systems to consider. This historical context sets the stage for a review of some of the more remarkable efforts to automate ID. Systems reviewed herein illustrate important lessons learned along the way. Consequently, the historical review of systems is not intended to be comprehensive or complete. Rather, it is designed to introduce key distinctions and to highlight what the instructional design community has learned through these attempts. The main theme of this chapter is that regardless of success or failure (in the sense of continued funding or market success), attempts to automate a complex process nearly always provide a deeper understanding of the complexities of that process.

See Appendix 1 for abbreviations used and Appendix 2 for glossary of key terms.

26.2 HISTORICAL OVERVIEW

One way to approach the history of ID automation would be to trace the history of automation in teaching and learning. However, this would take the discussion into areas outside the focus of this discussion, requiring a discussion of teaching machines (Glaser, 1968; Silverman, 1960; Taylor, 1972) among other forms of automation in teaching and learning. Rather than extend the discussion that far back into the twentieth century, the focus will remain on the latter half of the twentieth century and on automation intended to support ID activities.

Several researchers have pointed out that developments in instructional computing generally follow developments in software engineering with about a generation delay (Spector, Polson, & Muraida, 1993; Spector, Arnold, & Wilson, 1996; Tennyson, 1994). Some may argue that this is because ID and training development are typically perceived as less important than developments in other areas. A different account for this delay, however, is that educational applications are typically more complex and challenging than applications in many business and industry settings. Evidence in support of both accounts exists. The point to be pursued here is twofold: (a) to acknowledge that automation techniques and approaches in instructional settings generally follow automation in other areas, and (b) then to look at developments in other areas as a precursor to automation in ID.

Merrill (1993, 2001) and others (e.g., Glaser, 1968; Goodyear, 1994) have argued persuasively that ID is an engineering
discipline and that the development of instructional systems and support tools for instructional designers is somewhat similar to the development of software engineering systems and support tools for software engineers. Consequently, automation in software engineering serves as the basis for a discussion of automation in instructional design. What have been the trends and developments in computer automation in the field of software engineering?

To answer this question, it is useful to introduce the phases typically associated with a systems approach to engineering design and development. These phases include (a) analysis of the situation, requirements, and problem; (b) planning and specification of solutions and alternatives; (c) development of solutions or prototypes, with testing, redesign, and redefinition; (d) implementation of the solutions, and (e) evaluation, maintenance, and management of the solutions. Clearly these phases overlap; they are interrelated in complex ways; they are less discrete than typically presented in textbooks, and they are often accomplished in a nonlinear and iterative manner (Tennison, 1995). Although these software engineering phases become somewhat transparent in rapid prototyping settings, they are useful for organizing tasks that might be automated or supported with technology.

It is relevant to note that these software engineering phases may be regarded as collections of related tasks and that they correspond roughly with the generic 1D model called ADDIE—analysis, design, development, implementation, and evaluation. Additionally, these phases can be clustered into related sets of processes: (a) front-end processes such as analysis and planning, (b) middle-phase processes including design, development, refinement, and delivery, and (c) follow-through processes, including summative and formative evaluation, life cycle management, and maintenance. These clusters are useful for categorizing various approaches to automation. Goodyear (1994) clusters these phases into upstream and downstream phases, with the upstream phase including analysis and planning activities and the downstream phase including remaining activities.

Reviewing automation in software engineering, it is possible to identify a number of support tools for computer engineers and programmers. Syntax-directed, context-sensitive editors for coding were developed in response to a recognized need to create more readable and more easily modified programming code. Such editors improved the productivity of programmers in middle-phase activities (development and implementation) and had an impact on overall program maintenance in the life cycle of a software product (follow-through activities). In short, downstream activities in both the second and the third clusters were and still are supported with such tools.

More aggressive support for front-end and early middle-phase activities developed soon thereafter. IBM developed a flowchart-based language (FL-I and FL-II) that allowed a software engineer to specify the logic of a program in terms of a rigorously defined flowchart, which then automatically generated Fortran code to implement the flowchart specification. This was clearly a form of automation aimed at the intersection of the front-end and middle phases of software engineering, which suggests that the clustering of phases is somewhat arbitrary and that the phases, however clustered, are interrelated.

In the 1980s computer-assisted software engineering (CASE) systems were developed that attempted to integrate such tools with automated support for additional analysis and management tools so as to broaden the range of activities supported. These CASE systems have evolved and are now widely used in software development. CASE systems and tools provide support throughout all phases and address both upstream and downstream activities.

About the same time that code generators and syntax-directed editors were being integrated into CASE performance support systems, object-oriented systems developed. This resulted in the reconceptualization of software engineering in terms of situated problems rather than in terms of programming or logical operations, which had been the focus in earlier software development systems. This shift emphasized how people think about problems rather than how machines process solutions to problems. Moreover, in an object-oriented system, there is strong emphasis on a long-term enterprise perspective that explicitly addresses reuse of developed resources.

Whereas code generators actually replaced the human activity of coding with an automatic process, syntax-directed editors aimed to make human coders more efficient in terms of creating syntactically correct and easily readable code. The first kind of automation has been referred to as strong support, and the second type of system is called weak support (Goodyear, 1994, 1995; Halff, 1993; Spector, 1999). Strong systems are aimed at replacing what a human can do with something to be accomplished by a computer. Weak systems are aimed at extending what humans can do, often to make less experienced practitioners perform more like experts. Weak systems have generally met with more success than strong systems, although there are strong systems that are narrowly focused on a limited set of well-defined actions and activities and have met with success as well (Spector, 1999).

Automated support for the middle phases occurred first and was given primary consideration and emphasis. Automated support for front-end and for follow-through activities and processes have been less aggressively pursued and developed late in the evolution of the automation of software engineering processes. Integrated systems are now the hallmark of automation within software engineering and can be characterized as primarily providing weak support across a variety of phases and activities for a wide variety of users. The integrated and powerful performance support found in many CASE systems adopted tools and capabilities found in computer-supported collaborative work systems and in information management systems. These tools have now evolved into still more powerful knowledge management systems. Capabilities supported by a knowledge management system typically include (a) communications support for a variety of users; (b) coordination of various user activities; (c) collaboration among user groups on various project tasks and activities involving the creation of products and artifacts; and (d) control processes to ensure the integrity of collaborative activities and to track the progress of projects (Spector & Edmonds, 2002). Knowledge management systems can be found in a number of domains outside software engineering and represent a full spectrum of support across a variety of tasks and users.

This short review of automation in software engineering suggests several questions to consider in examining the automation
of ID and development processes.

1. Which phases are targeted for support or automation?
2. Is the type of automation intended to replace a human activity or to extend the capability of humans performing that activity?
3. Is how designers think and work being appropriately recognized and supported?
4. Are a long-term enterprise and organizational perspective explicitly supported?

Of course other questions are possible. We have chosen these questions to organize our discussion of exemplary automated ID systems because we believe that these questions and the systems that illustrate attempted answers serve to highlight the lessons learned and the issues likely to emerge as critical in the future. These four questions form the basis for the selection of systems that are examined in more detail. Before looking at specific systems, however, we discuss relevant distinctions, definitions, and types of systems.

### 26.3 DISTINCTIONS AND DEFINITIONS

To provide a background for our review of automated ID systems, we briefly discuss what we include in the concept of ID and what we consider automation to involve. We then identify the various characteristics that distinguish one system from another. These characteristics are used in subsequent sections to categorize various types of ID automation and also to provide a foundation for concluding remarks about the future of automated ID.

#### 26.3.1 ID

ID, for the purpose of this discussion, is interpreted broadly and includes a collection of activities to plan, implement, evaluate, and manage events and environments that are intended to facilitate learning and performance. ID encompasses a set of interdependent and complex activities including situation assessment and problem identification, analysis and design, development and production, evaluation, and management and maintenance of learning process and the ID effort (Gagné, Briggs, & Wager, 1992).

The terms instructional design, instructional development, instructional systems development (ISD), and instructional systems design are used somewhat ambiguously within the discipline (Gustafson & Branch, 1997; Spector, 1994). Some authors and programs take pains to distinguish ID from instructional development, using one term for a more narrow set of activities and the other for a larger set of activities. Most often, however, ISD is used to refer to the entire set of processes and activities associated with ID and development. ISD has also been associated with a narrow and outdated behavioral model that evokes much negative reaction. It is not our intention here to resolve any terminological bias, indeterminism, or ambiguity. Rather, it is our aim to consider ID broadly and to look at various approaches, techniques, and tools that have been developed to support ID.

The examination of automated support systems for ID largely ignores the area of instructional delivery, although authoring systems are mentioned in the section on types of support. There are two reasons for this. First, there are simply too many systems directed at instructional delivery to consider in this rather brief discussion. Second, the most notable aspect of automation in instructional delivery concerns intelligent tutoring systems and these systems have a significant and rich body of research and development literature of their own, which interested readers can explore. Our focus is primarily on upstream systems and systems aimed specifically at planning and prototyping, because these areas probably involve the most complex and ill-defined aspects to be found in ID.

It is worth adding that the military research and development community has contributed significantly to the exploration of automation within the domain of ID (Spector et al., 1993). Baker and O’Neil (2003) note that military training research contributed advances such as adaptive testing, simulation-based training, embedded training systems, and several authoring systems in the period from the 1970s through the 1990s. A question worth investigating is why the military training research and development community made such progress in the area of ID automation compared with the rest of the educational technology research and development community in that period.

#### 26.3.2 Automation and Performance Support

For the purposes of our discussion, a process involves a purposeful sequence and collection of actions and activities. Some of these actions might be performed by humans and some by machines. Automation of a process may involve replacing human actions and activities with those performed by a computer (non-human intelligent agent). As noted earlier, this kind of automation is referred to as a strong form of support. When automation is aimed at extending the capability of a human rather than replacing the human, the support is categorized as weak and the associated system is called a weak system. Weak systems in general constitute a form of performance support.

Job aids provide the most common example of performance support. A calculator is one such form of job aid to support humans required to make rapid and accurate calculations. Performance support may also involve paper-based items such as checklists or much more sophisticated computer-based support such as a tool that automatically aligns or centers items.

Performance support systems that keep hidden the rationale or process behind the decision or solution are referred to as black box systems. Systems that make much of the system visible to the user are called transparent systems. If users are not expected to acquire expertise, then a black box system may be more desirable and efficient. However, if users desire to acquire expertise or if they are expected to acquire higher-order capabilities, then a glass box may be preferable.

When a computer-based support system is embedded within a larger system it is generally called an electronic performance
support system (EPSS). An example of such a system is an aircraft maintenance system that includes an electronic troubleshooting guide that is integrated with the specific device status and history of the aircraft. Some EPSs provide intelligent support in the sense that they make at least preliminary decisions based on their assessment and diagnosis of the situation.

26.3.3 Intelligent Support Systems

Our definition of an intelligent system is derived from Rich and Knight (1991): Intelligent systems are those systems in which computers provide humanlike expert knowledge or performance. Early intelligent systems included those aimed at providing a medical diagnosis based on a preliminary review of a patient’s condition and a sequence of follow-up examinations aimed at isolating the underlying problem. Expert system technology of the kind used in diagnostic systems is only one form of artificial intelligence. Artificial neural networks represent another important category of intelligent systems; they have been used to recognize complex patterns and to make judgments based on the pattern recognized. Applications can be found in a number of areas including quality control and security systems. Intelligent systems may be either weak or strong.

Expert advisory systems are generally weak systems that extend or enhance the capability of a human decision maker. Intelligent tutoring systems are strong systems in that the burden for deciding what to present next to a learner is shifted entirely from a human (either the teacher or the student) to the instructional delivery system.

26.3.4 Collaborative Learning and Knowledge Management Systems

Additional characteristics that serve to distinguish systems are the number of users and the number of various uses. In software engineering, systems have evolved to support multiple users and multiple uses. A parallel development is beginning to occur with regard to automated support for ID. Given the growing interest in collaborative learning and distributed decision making, it is not surprising to find increasing interest in the use of multiple-user applications in various design and development environments (Ganesan, Edmonds, & Spector, 2001). This development is further evidence of the pattern reported earlier: advances in instructional computing are about a generation behind similar developments in software engineering.

26.3.5 Instructional Perspective

A final characteristic to consider is the issue of the underlying perspective or paradigm. This issue is more complex in the area of ID than in software engineering, where we have already noted the trend to adopt an object-oriented perspective. With regard to automated support for ID, there are additional perspectives to consider. Some of the prevailing instructional paradigms include constructionism (Jonassen, Hernandez-Serrano, & Choi, 2000), cognitive apprenticeship (Collins, Brown, & Newman, 1989), transaction theory (Merrill, 1995), and socially shared cognition (Resnick, 1989). The assumptions underlying these perspectives include the nature of knowledge, the learning environment, the role of the learner, and the role of the learner and instructional support. Does the system or support tool provide active and relevant support for a single versus a multiple learning paradigm or perspective? If software engineering continues to provide important clues about the future of ID technology, then the inclination will be toward flexible use and reuse, allowing for support of more than a single learning perspective or paradigm.

26.4 TYPES OF AUTOMATED ID SYSTEMS

Kasowitz (1998) identified the following types of automated ID tools and systems: (a) advisory/critiquing systems, (b) expert systems, (c) information management systems, (d) electronic performance support systems, and (e) authoring tools. Although these categories do overlap somewhat, they provide a reasonable organizational framework for considering automated ID systems developed in the latter half of the twentieth century.

26.4.1 Advisory/Critiquing ID Systems

The notion of an advisory critiquing system was introduced by Duchastel (1990). Duchastel proposed an advisory system that would be used to provide an ID team with a critique of a prototype or instructional solution given a set of desired outcomes and system goals. The system envisioned by Duchastel was never constructed, although an advisory system called PLANalyst created by Dodge (1994) did provide limited advisory feedback in addition to assisting in other planning activities. The lack of an advisory critiquing system reflects the complexity of such an enterprise. Such an advisory critiquing system would require sophisticated pattern recognition capabilities as well as a great deal of expert knowledge. Moreover, the prototypes and sample solutions provided would require some form of instructional tagging that has yet to be developed as well as access to extensive libraries of reusable learning objects (Wiley, 2001) and system evaluations and assessments (Baker & O’Neil, in press). Such an advisory/critiquing system remains a desirable long-term goal.

26.4.2 Expert ID Systems

In the latter part of the twentieth century, expert systems met with interest and success in various domains, including the domain of ID (Jonassen & Wilson, 1990; Spector, 1999; Welsh & Wilson, 1987). Some of these expert ID systems focused on specific tasks, such as generating partially complete programming problems in an intelligent tutoring system (van Merriëboer &
Paas, 1990) or automating the production of technical documentation for instructional and other systems (Emmott, 1998). Many such expert systems for focused tasks in ID can be found (Locatis and Park, 1992). Focused applications of expert system technology in general have met with more success than more general applications, although there were several notable developments of more ambitious expert ID systems in this period, including:

1. Instructional Design Environment (IDE; Pirrolli & Russell, 1990) — a hypermedia system for designing and developing instructional materials;
2. ID Expert (Merrill, 1998) — an expert system for generating instruction based on second-generation instructional transaction theory (which evolved into a commercial system called Electronic Trainer and influenced the development of XAIDA, which is described in more detail); and
3. DidoM (Gustafson & Reeves, 1990) — a rule-based, hypermedia system for instructional design and course development (which evolved into a system called ID Bookshelf for the Macintosh).

Among the applications of expert systems in ID are those that support the development of intelligent tutoring systems. van Merriënboer & Paas (1990) developed an intelligent tutoring system for teaching programming that included several rule-based systems to accomplish specific tasks, including the generation of partially solved programming problems. A wide variety of applications of expert systems within the context of intelligent tutoring systems is given by Regian and Shute (1992). Most of these are focused on the delivery aspects of instruction — creating a dynamic model of a learner’s understanding within a domain to generate a new problem to the learner. A remarkable exception to this use of expert systems within the context of intelligent tutoring was the Generic Tutoring Environment (GTE), which used an expert rule base and a robust instructional model to generate intelligent tutoring systems (Elen, 1998). GTE is elaborated in more detail in the next section.

26.4.3 Information Management and ID Systems

Information and knowledge management within the domain of ID have been largely based on other ID systems and developments as components and capabilities have been integrated and made interoperable (Spector & Edmonds, 2002). For example, although the expert, hypermedia system IDE is no longer in existence, the idea was to create an entire environment for instructional development (Pirrolli & Russell, 1990). Significant developments in this area have emerged from the cognitive informatics research group (LICEF) at Télé-université, the distance-learning university of the University of Québec. The LICEF research group consists of nearly a hundred individuals working in the fields of cognitive informatics, telecommunications, computational linguistics, cognitive psychology, education, and communication who have contributed to the development of methods, design and development tools, and systems to support distance learning (Paquette, 1992). This group has developed a range of tools that support the creation of a knowledge model for a subject domain, the development of a method of instruction for that domain, and the environment for the delivery of instruction in that domain (Paquette, Aubin, & Crevier, 1994). MOT, one of the knowledge modeling tools created by this group, is described in more detail in the next section.

26.4.4 EPSSs for ID

EPSSs are typically embedded within a larger application (e.g., an airplane) and provide targeted support to humans performing tasks on those larger systems (e.g., aircraft maintenance technicians). Within the context of ID, there have been commercial EPSSs (e.g., Designer’s Edge and Instructional DesignWare) as well as R&D EPSSs (e.g., DidoM). NCR Corporation commissioned the development of an EPSS for ID based on a development methodology called quality information products process (Jury & Reeves, 1999). Another example of an EPSS in ID is CASCADE, a support tool aimed at facilitating rapid prototyping within ID (Nieveen, 1999). An example of an EPSS for ID that is not tightly coupled with an authoring tool is the Guided Approach to Instructional Design Advising, which is described in more detail in the following section.

26.4.5 ID Authoring Tools

There has been a plethora of authoring tools to enable instructors and instructional developers to create computer- and Web-based learning environments (Kearsley, 1984). Early authoring systems were text based and ran on mainframes (e.g., IBM’s Instructional Interaction System and Control Data Corporation’s Plato System). Widely used course authoring systems include Macromedia’s Authorware and Click2Learn’s ToolBook. Many other course authoring systems have been developed and are still in use, including IconAuthor, Quest, and TenCore, which, along with other authoring languages, was developed from Tutor, the authoring language underlying the Plato System.

Specific languages have been developed to make the creation of interactive simulations possible. The creation of meaningful simulations has proven to be a difficult task for subject experts who lack specific training in the creation of simulations. The system that comes closest to making simulation authoring possible for those with minimal special training in simulation development is SimQuest (de Jong, Limbach, & Gelleijv, 1999). SimQuest includes a building blocks metaphor and draws on a library of existing simulation objects, making it also an information and knowledge management tool for ID.

The Internet often plays a role in instructional delivery and many authoring environments have been built specifically to host or support lessons and courses on the World Wide Web. Among the better-known of the commercial Web-based course management systems are BlackBoard, Learning Space, TopClass, and WebCT. Although there have been many publications about courses and implementations in such environments, there has been very little research with regard to effects of the systems.
on instruction. TeleTop, a system developed at the University of Twente, is a notable exception that documents the particular time burdens for instructors leading Web-based courses (Gervedink Nijhuis & Collis, 2003).

26.5 A CLOSER LOOK AT FOUR SYSTEMS

In this section we briefly describe a variety of automated instructional design systems, including the following:

- GAIDA (Guided Approach to ID Advising—later called GUIDE)
- GTE (Generic Tutoring Environment)
- MOT (Modélisation par Objets Typés)
- XAIDA (Experimental Advanced Instructional Design Associate—called an advisor in early publications)

26.5.1 GAIDA—Guided Approach to ID Advising

An advisory system to support lesson design was developed as part of the Advanced Instructional Design Advisor project at Armstrong Laboratory (Spector et al., 1993). This advisory system is called GAIDA. The system uses completely developed sample cases as the basis for helping less experienced instructional designers construct their lesson plans. GAIDA is designed explicitly around the nine events of instruction (Gagné, 1985). Gagné participated in the design of the system and scripted the first several cases that were included in the system while at Armstrong Laboratory as a Senior National Research Council Fellow (Spector, 2000). GAIDA allows users to view a completely worked example, shown from the learner’s point of view (see Fig. 26.1). The user can shift from this learner view to a designer view that provides an elaboration of why specific learner activities were designed as they were. The designer view allows the user to take notes and to cut and paste items that may be relevant to a lesson plan under construction.

GAIDA was also designed so that additional cases and examples could easily be added. Moreover, the design advice in GAIDA could be easily modified and customized to local practices. Initial cases included lessons about identifying and classifying electronic components, performing a checklist procedure to test a piece of equipment, checking a patient’s breathing capacity, handcuffing a criminal suspect, performing a formation flying maneuver, and integrating multiple media into lessons. GAIDA was adopted for use in the Air Education and Training Command’s training for technical trainers. As part of the U.S. government’s technology transfer effort in the 1990s, GAIDA became a commercial product called GUIDE—Guided Understanding of Instructional Design Expertise—made available through the International Consortium for Courseware Engineering with three additional cases.

As a commercial product, GUIDE was only marginally successful, although GAIDA continues to be used by the Air Force in the technical training sequence. The utility of this advising system is that it provides a concrete context for the elaboration of ID principles without imposing rigidity or stifling creativity. The user can select examples that appear to be relevant to a current project and borrow as much or as little as desired. Gagné’s basic assumption was that targeted users were bright (all were subject

---

![FIGURE 26.1. Adapted from screen from GAIDA/GUIDE.](image-url)
matter experts who had advanced to a recognized level of expertise in their fields) and motivated. All that was required to enable such users to produce meaningful lesson plans were relevant examples elaborated in a straightforward manner. Gaida/Guide achieved these goals. Users quickly advanced from a beginning level to more advanced levels of ID competence based on the advice and elaborated examples found in Gaida.

26.5.2 GTE—Generic Tutoring Environment

GTE grew out of an international collaboration involving academic and industrial communities in several countries and was focused on providing support for intelligent tutoring systems. GTE proceeds from a particular educational philosophy and explicitly adopts a particular psychological perspective involving the nature of expertise (Ericsson and Smith, 1991; Resnick, 1989). A cognitive processing perspective informs the design of GTE (van Marcke's, 1992a, 1992b, 1998). Van Marcke (1998), the designer of GTE, argues that teaching is primarily a knowledge-based task. Experienced teachers are able to draw on specific teaching knowledge in addition to extensive domain knowledge. A primary task for an intelligent tutoring system is to integrate that instructional knowledge in the system in a way that allows the system to adapt to learners just as expert teachers do.

van Marcke took an intentionally narrow view of the instructional context, confined instructional decision making to teachers, and did not explore the decisions that might be made by instructional designers, textbook authors, test experts, and so on.

GTE combines a reductionist perspective with a pragmatic approach. The tutor-generation task is reduced to two tasks: (a) determining all of the relevant components and relationships (an instructional semantic network), and (b) determining how and when to provide and combine these components to learners so as to promote learning (Fig. 26.2). The domain perspective in GTE consists of a static semantic network. According to van Marcke (1998), this network is used for sequencing material within a topic area, for indexing instructional objects, and for stating learning objectives. GTE makes use of an object-oriented network so that components can be meaningfully combined and reused.

Although a reductionist approach lends itself to automation in the strong sense, there are limitations. As van Marcke (1998) claims, (a) teaching is an inherently complex activity, (b) there are only incomplete theories about how people learn, and (c) strong generative systems should include and exploit expertlike instructional decision making. However, it is not completely clear how expert human designers work. Evidence suggests that experts typically use a case-based approach initially to structure complex instructional
planning tasks (Perez & Neiderman, 1992; Rowland, 1992). The rationale in case-based tools is that inexperienced instructional planners lack case expertise and that this can be provided by embedding design rationale with lesson and course exemplars. This rationale informed the development of GAIDA.

However, cases lack the granularity of the very detailed objects described by van Marcke (1998). A significant contribution of GTE is in the area of object-oriented instructional design. GTE aimed to generate computer-based lessons and replace a human developer in that process. GTE does not directly support student modeling in the sense that this term has been used in the intelligent tutoring literature, although van Marcke (1998) indicates that GTE’s knowledge base can be linked to student modeling techniques. GTE contains a number of instructional methods with detailed elaborations and basic rules for their applicability within a dynamic generative environment. When these instructional rules break down, it is possible to resort to human instructional intervention or attempt the computationally complex and challenging task of maintaining a detailed and dynamic student model. By not directly supporting student modeling, GTE remains a generic tool, which is both a strength and a weakness.

One might argue that it is only when a case-based approach fails or breaks down that humans revert to overtly reductionistic approaches. What has been successfully modeled and implemented in GTE is not human instructional expertise. Rather, what has been modeled is knowledge about instruction that is likely to work when human expertise is not available, as might be the case in many computer-based tutoring environments.

Because teaching is a complex collection of activities, we ought to have limited expectations with regard to the extent that computer tutors are able to replace human tutors. Moreover, it seems reasonable to plan for both human and computer tutoring, coaching, and facilitation in many situations. Unfortunately, the notion of combining strong generative systems (such as GTE) with weak advising systems (such as GAIDA) has not yet established a place in the automation of instructional design. We return to this point in our concluding remarks.

26.5.3 MOT—Modélisation par Objets Typés

MOT is a knowledge-based modeling tool aimed at assisting instructional designers and developers in determining what kind of content knowledge and skills are involved, how these items are related, and how they might then be sequenced for learning and instruction. MOT grew out of an earlier effort at Télé-université LICEF, a research laboratory for cognitive informatics and training environments at the University of Québec, to develop a didactic engineering workbench (Paquette, 1992; Paquette et al., 1994).

MOT allows a subject matter expert or designer to create a semantic network of a subject domain at a level of detail appropriate for instructional purposes (Fig. 26.3). The semantic network has two interesting features: (a) It is designed specifically for instructional purposes (e.g., there are links to indicate relationships that have instructional significance), and (b) the objects in the network are part of an object-oriented network (e.g., they can be instantiated at various points in a curriculum/course and retain relevant aspects).

MOT can be used as a stand-alone tool or in concert with other tools developed at Télé-université, including a design methodology tool (ADISA) and a Web-based delivery environment tool (Explor@). The suite of tools available provides the kind of integration and broad enterprise support found in other domains. This entire suite of tools can be regarded as a knowledge management system for ID (Spector & Edmonds,
26. Automating Instructional Design

ADISA embraces an instructional perspective that is similar to cognitive apprenticeship (Collins et al., 1989) and is actively supportive of situated learning (Paquette, 1996; Paquette et al., 1994).

MOT is a weak system in that it extends the ability of designers to plan instruction based on the knowledge and skills involved. The rationale in MOT is not as transparent as the rationale offered in GAIDA, which provides elaborations of specific cases. However, whereas GAIDA left the user to do whatever seemed appropriate, MOT imposes logical constraints on instructional networks (e.g., a user cannot create an instance that governs a process). Moreover, the object-oriented approach of MOT and its place in the context of a larger knowledge management system for ID has great potential for future developments.

26.5.4 XAIDA—Experimental Advanced Instructional Design Associate

Like GAIDA, XAIDA was developed as part of the Advanced Instructional Design Advisor project at Armstrong Laboratory (Spector et al., 1993). Whereas GAIDA explicitly adopted a weak approach to automated support, XAIDA aggressively adopted a strong approach with the goal of generating prototype computer-based instruction based on content information and a description of the learning situation provided by a subject matter expert or technical trainer. The underlying instructional model was based on ID2 (second-generation instructional design) and ID Expert (Merrill, 1993, 1998). A commercial version of ID Expert known as Electronic Trainer met with substantial success and the underlying structure is part of other systems being offered by Leading Way Technologies in California.

XAIDA was aimed at entry-level and refresher aircraft maintenance training (Fig. 26.4). In short, the domain of application was appropriately constrained and targeted users were reasonably well defined. As with the other strong system described here in GTE, such constraints appear to be necessary when attempting to automate a complex process completely. Whereas expert human designers can make adjustments to the many variations in domains, learners, and learning situations, a strong generative system cannot benefit from such expertise in ill-defined domains. Setting proper constraints is a practical way to address this limitation.

One of the more remarkable achievements of XAIDA was its linkage to the Integrated Maintenance Information System (IMIS), which consisted of two databases: One contained technical descriptions and drawings of the avionic components of a military aircraft, and the other contained troubleshooting procedures for those components (Spector et al., 1996). The basic notion of this innovation was to address a scenario such as the following. A technical supervisor has determined that an apprentice technician requires some refresher training on how to remove, troubleshoot, repair, and replace the radar in a particular aircraft. The supervisor goes to XAIDA-IMIS, selects the component about which just-in-need instruction is desired, and selects the type of training desired. XAIDA-IMIS then generates a module based on the current version of the equipment installed in the aircraft. IMIS has current information on installed equipment. Cases in technical training schools usually involve earlier versions of equipment. The XAIDA-IMIS module is specific to the need and to the equipment actually installed. The entire process of generating a just-in-need lesson required about 5 min—from the identification of the need to the delivered lesson.

Despite this remarkable demonstration of efficiency and effectiveness, the Air Force has since abandoned this effort. Nevertheless, the linkage to databases represents another extension of automated support into the domain of knowledge management for ID. Additionally, the requirement to constrain strong systems again demonstrates the limitations of automation within a complex domain.
These four unique systems are summarized in Table 26.1. The intention of this section was to illustrate a representative variety of ID automated systems so as to motivate a discussion of current trends and issues and to provide a foundation for speculation about the future of automation in the domain of ID.

### 26.6 RESEARCH FINDINGS

There has been considerable research conducted on these four systems as well as others mentioned earlier. What can be learned from the research on automated ID systems? First, evaluating automated ID systems is a complex problem (Gros & Spector, 1994). There are numerous factors to consider, including the type of system, the goals of the instruction developed, the ID team, and the instructors and learners for whom systems are created. Jonassen and Wilson (1990) propose a number of evaluation criteria similar to those developed for the evaluation of CASE tools. Montague and Wulfeck (1982) propose an instructional quality inventory to be used in evaluating instructional systems. Halff (1993) distinguishes three levels of evaluation for ID systems: quality review, formative evaluation, and summative evaluation. Halff also emphasizes the requirement to assure quality prior to conducting formative and summative evaluations.

Gayeski (1991) argues that an evaluation of automated ID systems requires consideration of uses by novice as well as expert designers and organizational considerations. In short, it is difficult to evaluate automated ID systems.

Most of the published research presents formative evaluations of systems or evaluations of learning environments created using particular systems. These research findings do not address the deeper issues associated with the four questions raised earlier, as it is difficult to link features of an ID system to improved learning and instruction or to longer-term trends in the development of learning environments. Two kinds of evaluation findings are worth noting. First, productivity improvements have occurred due to systems that provide performance support or automate portions of ID (Bartoli & Golas, 1997; Merrill, 1998; Spector et al., 1993). While results vary, using support tools can achieve an order of magnitude improvement in productivity of a design team. Second, learning outcomes can result from systems that enable designers to adapt systems to particular learning needs. The promise of intelligent tutoring systems was to raise learning outcomes by two standard deviations, similar to that noted for one-to-one human tutoring situations (Farr & Psotka, 1992). While such significant outcomes did not occur, there are many instances of significant improvement (as much as a standard deviation) in learning outcomes with regard to well-structured learning goals (e.g., beginning programming and simple troubleshooting) (Farr & Psotka, 1992; Regan & Shute, 1992). In addition to such findings, some evaluation findings with regard to the four systems described earlier are mentioned next.

Gaida has been evaluated in numerous settings with both novice and expert designers (Gettman, McNelly, & Muraida, 1999). Findings suggest that expert designers found little use for Gaida, whereas novice designers made extensive use of Gaida for about 6 months and then no longer felt a need to use it. GTE proved to be useful in generating intelligent tutors across a variety of subject domains, as long as the subject domains were sufficiently well structured (Elen, 1998). Mot has been used by novice and experienced designers for a variety of domains ranging from well-structured to ill-structured knowledge domains (e.g., organizational management). Paguette and colleagues (1994) found consistent improvements in both productivity (about an order of magnitude; similar to the productivity improvements of other systems) and quality (consistency of products and client satisfaction were the primary measures). Xaida was evaluated during every phase of its development (Muraida, Spector, O’Neil, & Marlin, 1993). Perhaps unique to the Xaida project was a serious evaluation of the design plan, along with subsequent evaluations of Xaida as it was developed. The final evaluation of Xaida focused on productivity and the results are again remarkable. As noted earlier, Xaida was linked to software to electronic databases that described aircraft subsystems and provided standard troubleshooting procedures for each subsystem. When Xaida was linked to these databases, a technical supervisor could generate a lesson for refresher training for an apprentice technician on a selected subsystem in less than 10 minutes (Spector et al., 1996).

We found no published research findings on the organizational impact of these systems, although anecdotal reports on nearly every system mentioned are easily found. Rather than review additional evaluation studies or present anecdotal evidence of the effects of these systems, we move next to a discussion of trends and issues likely to follow given what has already been accomplished with and learned from these systems.

### 26.7 TRENDS AND ISSUES

Although the attempts to automate ID are not by any means limited to the four systems outlined in the preceding section, we have used these systems for illustrative purposes. We believe that the serve as a representation of the major trends, issues, and possibilities that have been encountered in the process.

---

**TABLE 26.1: Automated Support for Instructional Design and Development**

<table>
<thead>
<tr>
<th>Type of Automation or Support</th>
<th>GAIDA</th>
<th>MOT</th>
<th>GTE</th>
<th>Xaida</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong or weak</td>
<td>Weak</td>
<td>Weak</td>
<td>Strong</td>
<td>Strong</td>
</tr>
<tr>
<td>Black box or glass box</td>
<td>Glass</td>
<td>Glass</td>
<td>Black</td>
<td>Opaque</td>
</tr>
<tr>
<td>Upstream or downstream</td>
<td>Up</td>
<td>Up</td>
<td>Down</td>
<td>Down</td>
</tr>
<tr>
<td>Single-user or multiple-user</td>
<td>Single</td>
<td>Group</td>
<td>Single</td>
<td>Group</td>
</tr>
<tr>
<td>Learning paradigm(s) supported</td>
<td>Multiple</td>
<td>Single</td>
<td>Single</td>
<td>Single</td>
</tr>
</tbody>
</table>

---

P1: MRM/FYX  P2: MRM/UKS  QC: MRM/UKS  T1: MRM
PB378-26  PB378-Jonassen-v3.cls  August 30, 2003  14:25  Char Count= 0
Two very reachable possibilities pertaining to efficiency and effectiveness of intelligent performance support for courseware engineering come to mind. The first concerns connecting object-oriented approaches with case-based advising, and the second concerns the creation of easily accessible, reusable electronic databases. The key to achieving both of these possibilities revolves around the key notions of object orientation, knowledge modeling, instructional tagging, learning objects, and instructional standards. These key ideas have been demonstrated in the systems here and exist in other systems as well.

First, let us consider connecting object-oriented approaches with case-based advising. Case-based advising has been demonstrated in GUIDE. Case-based advising could be made much more flexible if it were constructed within an object-oriented framework. This would mean that cases could be constructed as needed to suit specific and dynamic requirements rather than relying on prescribed cases, as found in GAIDA/GUIDE.

The notion of knowledge objects has emerged from object orientation in software engineering and the development of object-oriented programming languages such as SIMULA (Dahl & Nygaard, 1966). Basically the notion of object orientation is to think in terms of (a) classes of things with more or less well-defined characteristics or attributes, (b) objects that inherit most or all of the characteristics of a class and have additional built-in functionality that allows them to act and react to specific situations and data, and (c) methods that specify the actions associated with an object. A knowledge object might be considered as an instance within an information processing class that has the purpose of representing information or promoting internal representation and understanding. A knowledge object that is explicitly intended to facilitate learning is called a learning object (Wiley, 2001). Knowledge objects might be considered the building blocks of a knowledge construction set within an instructional system, although this metaphor should not be taken literally or casually.

The general notion of object orientation is twofold: to promote analysis and problem solving in terms that closely parallel human experience rather than in terms that are tightly linked to machine processing features, and (2) to promote reuse. Object orientation was initially conceptualized in terms of improved productivity, although there has been a clear shift toward viewing object orientation in education as being aimed primarily at improving understanding. The value of knowledge objects in promoting organizational learning is recognized in many knowledge management systems.

Second, dramatic improvements in development time and cost are now a reasonable goal. An object-oriented approach allows instructional objects to be constructed dynamically and flexibly, as ably illustrated by GTE (van Marke 1992a, 1998).

The temptation will most likely be to base strong automated support for ID on knowledge objects and metatagging (Table 26.2). Identifying a sufficient set of instructional tags and then devising a facile way to support tagging of existing and new electronic databases are a significant and challenging undertaking but would be eminently worthwhile.

There is an active effort to create a standardized extensible markup language called XML (Connolly, 1997). This language is similar to HTML but is intended to provide a syntax for defining a specialized or customized markup language, returning to the original notion behind SGML (Standard Generalized Markup Language) with the advantage of a decade of experience. Basically, XML is a low-level syntax for creating new declarative representations for specific domains. Several such instantiations have been developed including MathML for mathematical expressions and SMIL for scheduling multimedia presentations on the Internet. A quite natural research and development project that could be associated with the XML effort would be to create, implement, and evaluate an instructional markup language using XML as the underlying mechanism.

Clearly, the use of object-oriented approaches makes it possible in principle to reuse previous courses, lessons, databases, and so on. Two long-range goals come to mind. One has to do with connecting object-oriented design with case-based advising and guidance (for learners, instructors, and designers). Case-based advising could be made much more flexible if it were constructed within an object-oriented framework. Cases could be constructed from collections of smaller objects and could be activated according to a variety of parameters. Thus, cases could be constructed as needed to suit specific and dynamic requirements rather than relying on prescribed cases or a case base.

Both object orientation and case libraries are making their way into commercial authoring products as well. For example, PowerSim, an environment for creating system dynamics-based systems, has tested the ability to provide model builders with partially complete, preconstructed generic structures with all the relevant properties of reusable objects (Gonzalez, 1998). Such reusable, generic structures (adaptive templates) will most likely appear in other commercial systems as well. Creating reusable objects and case libraries makes more poignant the need for advising those who must select relevant items from these new ID riches.

Knowledge management systems developed somewhat independently of object orientation in software engineering. They have evolved from early information management systems that were an evolution from earlier database management systems. Databases were initially collections of records that contained

<table>
<thead>
<tr>
<th>Notional Instructional Tag</th>
<th>Instructional Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_definition</td>
<td>Identify a key definition, automatically generate of glossary entries</td>
</tr>
<tr>
<td>good_example_of</td>
<td>Highlight an exemplifying item, generate of an introductory example or reminder item</td>
</tr>
<tr>
<td>non_example_of</td>
<td>Emphasize a boundary case or exception or contrasting example, generate of an elaboration sequence</td>
</tr>
<tr>
<td>bad_example_of</td>
<td>Highlight an important distinction, generate of an elaboration sequence</td>
</tr>
<tr>
<td>moral_of_story</td>
<td>Summarize a main point, generate of a synthetic sequence</td>
</tr>
<tr>
<td>theme_of_article</td>
<td>Provide a very short abstract sentence, generate of an introductory sequence</td>
</tr>
<tr>
<td>main_point_of_paragraph</td>
<td>Summarize a short module or sequence, generate of a remedial or refresh sequence</td>
</tr>
</tbody>
</table>
individual fields representing information about some collection of things. Early databases typically had only one type of user, who had a specific use for the database. As information processing enjoyed more and more success in enterprise-wide situations, multiple users became the norm, and each user often had different requirements for finding, relating, and using information from a number of different sources. Relational databases and sophisticated query and user access systems were developed to meet these needs in the form of information management systems. As the number and variety of users and uses grew, and as the overall value of these systems in promoting organizational flexibility, productivity, responsiveness, and adaptability became more widely recognized, still more powerful knowledge management systems were developed. Knowledge management systems add powerful support for communication, coordination, collaboration, and control of information management systems and generally make use of some kind of fundamental object orientation system.

26.8 CONCLUDING REMARKS

Attempts to automate aspects of ID have led to deep insights into the more general processes of ID and development. Indeed, the theme of this chapter is that we have only begun to witness the many ways in which intelligent support can be provided to instructional designers and developers. However, we would be wrong to expect computers to replace human expertise in crucial areas of ID and development.

In the process of attempting to automate ID, key lessons include the following:

1. Strong systems work only in narrow and well-defined domains; there will continue to be opportunities to develop intelligent agents in support of learning, more so in the delivery domain (downstream activities) and less so in the planning and analysis phases (upstream activities).
2. Knowledge management systems are by nature weak systems and have a definite place in ID, which is by nature complex and often involves teams and enterprise-level learning and performance issues; some components of a knowledge management system for ID may involve strong and intelligent support, as in an intelligent agent to perform a particular task.
3. The value of knowledge objects and reuse is likely to be realized only when humans are kept involved and systems kept open.
4. We learn a lot about a process by trying to automate it; we find out how much human involvement is required and when and why such involvement is advisable. We should not discourage or denigrate attempts to automate more ID processes—we always learn something in the process.
5. The temptation will be to base strong automated support for human knowledge work on intelligent agents and metatagging.
6. Human involvement in the ID process will still be necessary and the real value of reusability and knowledge management in support of ID will be realized only if humans are involved in the process (Ganesan et al., 2001).

In conclusion, it seems safe to say that the future of intelligent performance support for courseware engineering will be filled with interesting opportunities and challenges. The most interesting opportunities may arise in inter- and multidisciplinary settings, which appear to be essential to making significant advances. The big challenge will be to remain humble with regard to how little we really know about human learning and intelligence.

As we develop new and more sophisticated techniques for creating learning environments, we should remember that learning cultures are dynamic. When only mainframe, text-based systems were available, users and learners came to expect certain types of screens and interactions. Today’s learners are more accustomed to highly interactive, multimedia environments, often attached to the World Wide Web. Learning environments that do not take these expectations into account may fall short of their intended effectiveness. Moreover, humility should remind us that machines are not likely to replace all aspects of human intelligence and expertise in the area of ID and development.

APPENDIX 1. ABBREVIATIONS AND ACRONYMS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIDA</td>
<td>Advanced Instructional Design Advisor</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer-assisted software engineering</td>
</tr>
<tr>
<td>EPSS</td>
<td>Electronic performance support system</td>
</tr>
<tr>
<td>GAIDA</td>
<td>Guided Approach to Instructional Design Advising</td>
</tr>
<tr>
<td>GTE</td>
<td>Generic Tutoring Environment</td>
</tr>
<tr>
<td>GUIDE</td>
<td>Guidance for Understanding Instructional Design Expertise</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>ID</td>
<td>Instructional design (broadly conceived; sometimes also called instructional development)</td>
</tr>
<tr>
<td>ID²</td>
<td>Second-generation instructional design</td>
</tr>
<tr>
<td>IDE</td>
<td>Instructional Design Environment</td>
</tr>
<tr>
<td>IMS</td>
<td>Integrated Maintenance Information System</td>
</tr>
<tr>
<td>ISD</td>
<td>Instructional systems development</td>
</tr>
<tr>
<td>IPSS</td>
<td>Intelligent performance support system</td>
</tr>
<tr>
<td>KQML</td>
<td>Knowledge Query Markup Language</td>
</tr>
<tr>
<td>MOT</td>
<td>Modélisation par Objets Types (a knowledge modeling tool)</td>
</tr>
<tr>
<td>SCORM</td>
<td>Shareable Content Object Reference Model</td>
</tr>
<tr>
<td>SGML</td>
<td>Standard Generalized Markup Language</td>
</tr>
<tr>
<td>XAIIDA</td>
<td>Experimental Advanced Instructional Design Associate</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>

APPENDIX 2. GLOSSARY OF KEY TERMS

**Advisory system**: In instructional design (ID) a computer system that implements principles and rules (analyze, design, develop, implement, evaluate) in guiding the developer through the instructional design process.

**Black box system**: Performance support systems that keep hidden from the user the rationale behind the decisions being made and the processes performed.
courseware engineering: An emerging set of practices, tools, and methodologies that result from an engineering approach to instructional computing systems. An engineering approach is in contrast to a craft or artisan approach and emphasizes the use of principled methods rather than intuition; an engineering approach values the replicability of processes and results rather than idiosyncratic creativity.

Downstream support: Support for implementation-intensive activities, such as the production of graphics, during the courseware development process.

Electronic performance support system: An interactive computer-based environment or infrastructure, embedded within a larger system, that provides support to facilitate performance. This includes capturing, storing, and distributing knowledge, information, advice, and learning experiences with minimum support from other people.

Expert system: A computer-based representation of the domain-specific knowledge of an expert in a form that can accessed by others for problem solving and decision making. An expert system typically contains a set of rules, a way to represent various situations, and an inference engine to determine which rule to activate given a particular situation.

Generative approach: Systems that take over the more routine aspects of ID and accelerate the process of the production of instruction, leaving the more conceptual aspects of ID to the designer.

Intelligent systems: Those systems in which computers provide humanlike expert knowledge or performance. Expert system technology represent one type of intelligent systems; artificial neural networks represent another category of intelligent systems.

Knowledge management systems: An evolution from earlier information management systems in which support for a variety of users is included. This leads to power support for communication, coordination, collaboration, and control on top of information management systems.

Knowledge objects: A way to organize a knowledge base so that different instructional algorithms can use the same knowledge objects to teach the same subject matter content.

Learning object: Any entity, digital or nondigital, that can be used or referenced in technology-supported learning.

Metadata: Data about data that describe something about the data intended to promote reuse and usability in general. In the past, metadata have largely been of concern to programmers and specialists; however, as the number of learning objects continues to increase, creators of learning objects should also create metadata categories and values.

Multiple-paradigm support: The ability to support more than one instructional perspective in accordance with an analysis of features of the situation at hand.

Multiple-user support or groupware: Various techniques that allow users to share documents, media, and other items in a database, exchange remarks, and create alternative versions of these items.

Object orientation: A way to think in terms of (a) classes of things with more or less defined characteristics, (b) objects that inherit characteristics of classes and have a built-in functionality that allows them to react to specific situations and data, and (c) methods that specify the actions associated with the objects.

Process automation: Automation of a process involves replacing human actions with those performed by a computer or nonintelligent agent.

Strong support: A performance support system that intends to replace part of an activity or process previously performed by a human with an automatic process is characterized as a strong system.

Transparent systems or glass box systems: Performance support systems that provide an explanation and make much of the systems’ reasoning evident to the user.

Upstream support: Support for analysis and design-intensive activities, such as definition of the target audience, information to be learned, learning goals, and objectives.

Weak support: A performance support system that intends to extend the capability of one or more humans performing various actions and activities is characterized as a weak form of automation or a weak system.

References


