Tech., Inst., Cognition and Learning, Vol. 2, pp. 249-260
Reprints available directly from the publisher
Photocopying permitted by license only

DO WE NEED AUTHORING SYSTEMS? A COMMERCIAL PERSPECTIVE

WELLESLEY R. FOSHAY* AND FRANK PREESE

PLATO Learning, Inc

The basic cost of development for the various types of e-Learning has not changed much in a generation. Authoring systems for e-Learning in all its forms has not progressed as quickly as general-purpose software engineering tools. As a result, most e-Learning today is created using general-purpose tools, the types of e-Learning products created are limited by the costs of development, and productivity gains have been largely limited to creation of media assets. To provide productivity gains, authoring systems need to address all phases of development and maintenance, they need to be designed for use by large, matrix-managed teams, and they need to meet the needs of large-scale production coding environments. Each of the major strategies for tool development has tradeoffs in terms of these requirements.

Keywords: e-Learning Authoring, e-Learning development productivity

The good news is that the explosive growth in e-learning over the last decade has led to many more authors creating much more e-learning content. The other piece of good news is that there are a great many more instructional models for e-learning in common use than was true even as late as 1990. Back then, the dominant model was direct tutorial instruction, a highly interactive emulation of a "Socratic Dialog." Today, it appears that proportionately little of the e-learning content produced uses the tutorial

*Corresponding author: rfoshay@plato.com

model, even in its theoretically current form. Most of what is being produced now appears to draw inspiration from the classroom lecture and the textbook, rather than the frequent interaction and feedback of well-designed computer tutorials. Online content presentation in the majority of examples appears to be akin to assigned reading, with meaningful interaction restricted to later e-mail or chat room exchanges. Other instructional models (such as project-oriented "WebQuests" and various kinds of interactive problems, simulations and games) are found occasionally, but appear to be relatively rare in use – at the very time that access to computing has become nearly ubiquitous.

The emergence of easy to use tools for creating all types of digital content has all but eliminated most specialized e-learning authoring systems. For a generation, such systems were common. For example, descendants of the PLATO system's original mainframe language, TUTOR, include over a dozen authoring systems, most recently Macromedia's Authorware. However, these systems did not transition well to the contemporary browser-based platform (presumably because sales were too small to justify the investment). Most e-learning content is now created with a variety of general-purpose HTML/XML web authoring tools, perhaps supplemented by Flash or Java. Specialized e-learning authoring tools in common use today, such as those included in the Blackboard platform, have less to do with instructional models and interactivity and more to do with integrating with the platform, publishing and communication. Today's general purpose tools do little to facilitate systematic, theoretically-grounded instructional design and development. The marketplace appears to have concluded that the benefits gained by using a specialized authoring tool to facilitate the processes of instructional design at any phase of the e-learning creation process do not exceed the drawbacks of the time required to master such tools and the additional complexity for deploying the work product to a large audience. Thus, while there is a lot more e-learning content now, it is being made with tools which are instructionally "dumber" than those of a decade ago. Ironically just when the instructional state of the art has progressed, little of that progress is reflected in the tool sets in broad use. In effect, we have regressed instructionally. We are over a generation into the age of e-learning, but only a decade into the age of the Internet. Perhaps people don't change as fast as computers, and e-learning is still in the uncritical "gee whiz!" phase of early technology adoption.

We should ask if the current status quo is really satisfactory. Are there unmet needs which better authoring tools might cost-effectively meet? Our

perspective on the question is that of PLATO Learning, Inc., a large (over $140-million) enterprise with a 40-year e-learning heritage. In a typical year the company produces over 400 hours of e-learning products using a variety of models. These models include tutorials, intelligent tutors, instructional simulations and games, supplemental multimedia informational products, assessments, collaborative learning environments and knowledge bases, for use by our clients and internally. We maintain a library of over 4,000 hours of e-learning products, most delivered via the Web as tens of millions of student-hours of learning per year, world-wide.

To decide if there is really a need for e-learning authoring systems, it is useful to consider four issues: the economics of e-learning development, the dynamics of development teams, the needed instructional models, and the technical requirements of a large-scale commercial production and delivery environment. We will address each of these issues next.

THE ECONOMICS OF E-LEARNING AUTHORING

One of the biggest limiting factors on the growth of e-learning has been the cost of development. Our experience is that tutorials require around 200 hours of development effort to produce one hour of completed product; simulations, games, and intelligent tutors require between 200 and 800 hours per hour of completed product to produce. Less interactive presentation models typically require up to 100 hours per hour of instruction to produce. Development ratios as low as 40:1 have been reported for some kinds of resource creation which limit themselves to on-line text and the occasional drawing, with interaction confined to chat facilities during delivery. Furthermore, our experience is that these cost ratios have not substantially changed in a generation.

While the cost ratios have not changed materially, the production values have improved substantially. As the media capabilities of the e-learning tools have improved, graphics, audio, video and advanced text designs have been used to engage the learner and facilitate instruction. The biggest gains in productivity for e-learning have been in media production, made possible by the current generation of general-purpose media production tools. The market has shown us that more media-rich products are required to be competitive. Producing these products can be done for very little additional cost compared to a decade or more ago.

With the authors in this issue, we wonder what potential there is for

productivity gains elsewhere in the e-learning development process. It is interesting to begin examination of this question by seeing how the costs of development array over the phases of the development process. While practices vary by product type and producer, in our experience development costs up to release to Beta Test are distributed roughly like this:

1. Front-end content/audience analysis: 20%
2. Instructional strategy definition, rapid prototyping, scripting and storyboarding: 30%
3. Creation of media and software assets: 20%
4. Integration of components and testing: 30%

It should be clear that the improvements in productivity offered by current-generation general-purpose production tools affect only Step 3 and part of Step 4, or roughly 30% of project cost. In other words, up to 70% of production cost is not addressed by current general-purpose web authoring and learning management tools. This should make it clear why the costs of development have not changed much in a generation, and why productivity gains have been essentially limited to increased use of and better integration of media. It is also interesting that the current e-learning authoring tools are adept at packaging assets but do little to facilitate testing of either software integrity or instructional designs. In order to gain productivity in testing, a producer will turn to general-purpose software testing tools, but these help only with software integrity. A significant investment has to be made in these tools before real productivity gains will be seen. As a result, the small gains in productivity have not resulted a net reduction of e-learning development cost.

There is an important Phase 5 in most products: maintenance and upgrade, whose costs can be substantial. In a typical PLATO product life cycle, products will be upgraded as a result of instructional improvements, localization to additional markets, development of derivative products, platform changes, upgrades to media assets and interfaces, bug/error remediation, and content changes. Since PLATO products often have a life cycle of a decade or more, it is likely that all these modifications will approach or even exceed the initial cost of development.

E-learning producers have reacted to this cost structure by using three strategies. The first strategy has been to limit use of highly interactive product types. For the most part, the highest-cost types of e-learning (tutorials, intelligent tutoring, simulation and games) have been largely limited to the few applications where there are a very large number of

learners (and cost per learner is thus low), or when the need is of sufficiently high criticality to justify a high cost per learner – and the consumer is sufficiently sophisticated to value such costly products. An example of the former case is a typical PLATO course, which will serve millions of school and adult learners over its life span. An example of the latter case is a pilot training simulation, where lives depend on effective learning by a small number of learners, and alternative means of training (in real planes or fully realistic flight simulators) are unacceptably dangerous or costly.

The second strategy has been to emphasize use of the lower-cost e-learning product types, such as online text publishing with limited interaction, even when these entail higher cost later on (in effect, by moving most interaction costs to the delivery phase). Examples here include the great majority of "virtual college" or "virtual school" distance learning applications – delivered, for the most part, to relatively small numbers of learners who do not value (or are unaware of the value of) the more interactive, higher-cost e-learning options. This is the primary market for most of the learning management systems for distance learning.

The third strategy has been to reduce costs of the third and fourth phases of development (media asset production, integration and testing) through use of standardized templates and reusable code (ranging from standard objects for learning management integration and bandwidth management to clip art to templated interaction sequences). Already mentioned is use of tools to automate certain repetitive tasks occurring in media creation and software testing.

Interestingly, nearly all current, commercial general-purpose web development tools combine the second and third strategies. Most e-learning creation tools have focused on the first and second strategies, but have done relatively little with the third. PLATO Learning and some other large-scale developers have attempted to streamline development of the high cost types of e-learning, by using proprietary tools for limited purposes and well-defined repetitive tasks.

If we look at this trend by phase of development, it becomes clear that commercial web authoring tool development has focused almost exclusively on the third phase of development – roughly 20% of initial development cost. Proprietary and third-party tools have sometimes attempted to automate some of the repetitive tasks in the fourth phase – the last 30% of development cost. But again, these efforts have been largely limited to the low-cost product types. There have been very few recent commercial attempts to automate any phase of the production of complex product types such as tutorials, intelligent tutors, instructional simulations

and games (though tools for non-instructional application are available). There have been very few attempts to automate the first two phases — 50% of development cost— for any product type. Furthermore, maintenance and upgrade has been mostly ignored. In the general world of software engineering, enterprise-grade development tools go to great lengths to ease the cost of maintenance and upgrade; this is rarely true of e-learning authoring tools. We conclude that commercially available tools in common use today for general web authoring or for e-learning are missing a great many opportunities for automation and that is why there has been so little progress in reducing the costs of e-learning development, especially for the complex product types.

What is missing are tools which automate and integrate as much as possible of all four phases of initial development, as well as maintenance and upgrade, and do so for the highest-cost/value product types. Unfortunately, there probably are not enough producers of the high-cost product types to make such tools commercially viable on the open market, so these tools will probably continue to be proprietary or otherwise developed privately on a non-commercial, research or one-off project basis. For the most part, the tools described in this issue seem to concentrate on the second, or first and second, phases of the process; and they do so for the higher-cost product types. We view these efforts as urgently needed and very promising, but their productivity gains are mostly unproven in commercial production environments.

In general, since the tasks of the last two phases of initial development are the most standardized and repetitive, it is most likely that these tasks are best suited to automation through tool development. These tools are likely to have the broadest generality across a range of product types. However such gains will do little to improve the instructional design of the e-learning created by these authoring tools unless the producer makes a concerted effort to do so. Thus it may be that the biggest short-term gains in productivity will come in the last 50% of the e-learning development process, rather than the first 50% where the tools in this issue are focused.

THE DYNAMICS OF DEVELOPMENT TEAMS

In a typical large-scale development organization, development processes are very non-linear, and they are performed by large teams with varying roles and skills specialization (often a dozen or more roles, in teams

of up to 100 in size). The teams are likely to be geographically dispersed, so in a real sense the team resides on the global corporate network. The products are complex, encompassing thousands of files in multiple versions. To be of use on a large-scale commercial project, an authoring tool must be designed to support this kind of work environment. Tools of this sort do exist in software engineering and content management environments, but they are not as common in e-learning authoring.

For commercial development there are at least six implications for an authoring tool environment . First, the authoring tool must allow for task specialization corresponding to the varying roles of the team. Second, the tool should allow workflow and resource assignment to be parallel, flexible and manageable. For example, an authoring tool might allow the user to define a number of "work stations" or views, for different team members. Typical work stations might include:

Project Manager
Product Manager or client
Subject Matter Experts
Content Reviewers
Instructional Designers
Instructional design reviewers
Writers
Graphic designers
Graphic artists
Paste-up specialists
Programmers
Software testers
Students/users
Instructors/users
Archivists/librarians

Third, work products should be highly modular, to allow manageable, defined tasks and flexibility in workflow and resource assignment. Fourth, work flow must be clearly defined and tracked. In our experience this is done outside of the authoring tool using configuration management tools, IRC, Wikis, project documentation and the like. In the ideal authoring tool there would be features supporting defined handoffs, parallel task structures, and version control (so everyone is starting with the correct version of the work product) features supporting dependency checking between work products (so, for example, the graphic designers for a given module are assured of working from that module's storyboards), configuration

management, project management reporting, work flow monitoring, task scheduling and team communication. Fifth, the tool must support quality assurance processes. There could be features for defining quality standards for each task, inspection and testing, debugging, defect tracking, work product promotion and acceptances. Sixth, for products which automate the first two phases of the development process, task definitions and tool capabilities should support rapid prototyping, user trials, and iterative development.

INSTRUCTIONAL MODELS

Given the wide range of e-learning product types (tutorials, intelligent tutors, various flavors of simulations and games, assessment systems, and information presentations, any of which may include any level of interaction and collaboration), it is probably unrealistic to envision tools which support more than one of them. Even within product types, however, there is a considerable difference of opinion over how much structure the tool should impose, and thus, what instructional model (or range of models) should be embedded in a given tool. For example, as Murray points out in this issue (see also Murray, 2003, p.518), there is no single definition of what an instructional simulation is, what the design tradeoffs are for various features and types of simulations, or what design processes are needed to create various simulation structures.

Four strategies have been used to automate instructional model creation: templating, objects, design languages and model-based design. We will discuss each strategy.

If real design tools are hard to envision, perhaps the most common automation strategy is to emphasize reusability of code through templates, which capture particular outputs of specific design decisions. For example, templates in a tutorial authoring tool will probably be based on assumptions about how to do explanation, questioning, answer analysis and feedback. There may be variations of templates for different knowledge types. There also may be templates with models for varying types of practice and testing. And, there may be templates embodying models of branching based on a theory of learning, which may vary by type of knowledge (so that varying types of declarative and procedural knowledge use different templates, as an example). There are examples of similar families of templates with comparable structures for simulation and game authoring, and even for on-

line explanatory text/graphic displays.

Most templating tools reflect some particular design theory. However, this is a two-edged sword: on the one hand, it is precisely such theories which allow the template designers to standardize authoring tasks enough to create the templates. On the other hand, design theories are far from standard, and design problems and their solutions vary. But building in this kind of flexibility adds considerable complexity to the templates, and may diminish their value as a productivity aid. We suspect that the templating strategy works best when it can be relatively inflexible, by keying it to the needs of a specific, proprietary product type. The more generalizable the templating tool is, the more likely that the resulting templates will be too "generic" or complex to be satisfactory. Perhaps it is more realistic for a general-purpose authoring tool to provide facilities for creating and importing templates which reflect the requirements of each author and shared within a community of practice.

Another approach to reusability is to create libraries of objects. Object-oriented programming is now familiar, and there has been a great deal of interest in analogous content creation strategies surrounding "learning objects." In our experience, however, learning objects suffer many of the same limitations as templates: the more capable the object, the greater the level of complexity and the harder it is to understand, lessening its potential for reuse. Thus we believe that the greatest potential for learning objects as an automation strategy may be in proprietary, well-defined and highly structured contexts – the opposite of the claims for this strategy.

The third instructional model automation strategy is to aim for generalizability by basing the tool on a design-oriented language. This is the oldest automation strategy, going back to the original mainframe days of e-learning. Traditionally, these languages have been intended for maximum flexibility, but have offered little support for explicit instructional theory. Examples include PLATO's mainframe TUTOR language and most of its descendents. Recently, Gibbons has proposed development of a design language for instructional simulation (Gibbons, Nelson et al. 1999). This design language would embody a particular theoretical framework, but would retain much of the generalizability and flexibility of any language, making it useful for a wide range of instructional simulation types. As with the template-based tools, however, the two-edged sword of standardization applies to languages: the more flexible the language, the wider the range of uses it can accommodate. However, the very same flexibility adds complexity, which diminishes the ability of the tool to improve productivity.

A fourth automation strategy is to use the concept of patterns and model based design from the practice of software engineering. Patterns embody the experience of designers in solving problems which occur frequently. The authoring tool could provide a library of instructional patterns and documentation about what problem each pattern is designed to solve. If the authoring tool contains functions for creating models, it could recommend patterns by inspecting the model that author is creating. Organizations could create their own patterns and add them to the pattern library building up a proprietary repository of best practice design solutions.

Compromises are possible. Perhaps the optimum strategy for automating instructional models should include some kind of meta-language for custom template or object development, and the ability to create "production" work environments based on these custom templates or objects. At PLATO Learning, proprietary tool development has followed exactly this path: the foundation classes of objects are built in a general-purpose web authoring language (Flash), but the language is used to build specialized "editors" which are used to fill in Flash templates, thus creating particular kinds of objects defined by the underlying theories of knowledge structures and instruction as well as a variety of other product type parameters. A given collection of "editors" trades away generalizability for efficiency, but new "editors" can be created to meet new needs.

TECHNICAL REQUIREMENTS OF LARGE-SCALE DEVELOPMENT

Large-scale e-learning development is a special case of large-scale software development, and any authoring tool needs to provide for, or accommodate, the requirements of such development environments. Already described is the importance of systems with defined roles and tasks, modular deliverables, work flows, and facilities for work flow management version and quality control of every task.

In addition, the tool should be designed to support a current-generation software development methodology. Such methodologies emphasize rapid prototyping, non-linear design, user involvement in design, total quality management, automated and structured testing, and a staged development process with a progression of go/nogo decision points for the project.

Should an authoring tool be powerful enough to accomplish all of the above, it still needs to package the results in a form that can be delivered to users.

Today the expectation is that users will access the e-learning content through a browser. Content delivered through a browser involves server processing and workstation processing. Precisely how much work is required of the server and the workstation is an important matter of the design of the system and will directly impact the scalability of the product and the diversity of workstation environments that are supported. This is an area which challenges many authoring tools. No matter how sophisticated the instructional design and development capabilities of the tool, it must be able to output assets in a form which meets the delivery requirements of the product.

In a web environment such as PLATO Learning's, the "production" hardware/system platform must be scalable to support many thousands of simultaneous users using a variety of workstation environments to access the instruction. Any authoring tool must of necessity be designed for a particular "target" production platform and target workstation platforms, and the characteristics and limitations of that platform will play an important role in determining what the capabilities of the authoring tool will be. Unfortunately, the things which run most efficiently often represent the worst instructional design practices.

Finally, the tools must have commercial longevity. Users of the tool must be assured that the tool will be available, and current, for the full life cycle of the e-learning products they create. For some users, the product life cycle is only a few months. For others, it can be a decade or more, with frequent periodic maintenance and upgrade a major requirement. Over that long a period, there will likely be platform changes, ranging from new release levels of browsers and system components, to new hardware platforms, operating systems and network protocols. The tool and the company behind it must be capable of supporting the expected rate of change.

CONCLUSION

Compared to general software development, the automation of e-learning development is in its infancy, even after a generation of experience. The tools discussed in this issue represent promising examples of some of the strategies described here. Even though none of the tools represent a complete solution for all types of commercial e-Learning development, a number of them are potentially useful for automating certain development tasks for certain kinds of products. Their ultimate value will depend on their ability to automate the most labor-intensive parts of development, if only for

a particular product type. If they can be adapted to work harmoniously with a range of other tools aimed at automating other tasks, then significant productivity gains may result in the form of reduced cost, higher quality, or improved time to market. Thus we urge the reader to gauge the value of these tools, not as comprehensive solutions, but as components of potentially powerful, larger solutions.

REFERENCES

Buschmann, Meunier, Rohnert, Sommerlad, Stal (1996). Pattern-Oriented Software Architecture: A System of Patterns. Wiley & Sons

Gibbons, A. S., J. S. Nelson, et al. (1999). Model-Centered Analysis Process (MCAP): a pre-design analysis methodology. White Paper. Logan, UT**:** 38.

Murray, T. (2003). An Overview of Intelligent Tutoring Authoring System Tools: Updated Analysis of the State of the Art. In Murray, Ainsworth & Blessing (eds.) *Authoring Tools for Advanced Technological Learning Environments*. The Nethelands: Kluwer, pp. 493-546.