Evaluation of Two Intelligent Tutoring System Authoring Tool Paradigms: Graphical User Interface-Based and Text-Based

Shrenik Devasani  
Stephen B. Gilbert  
Virtual Reality Applications Center  
Iowa State University  
1620 Howe Hall, Ames, IA 50011.  
515-294-3092  
shrenik@iastate.edu, gilbert@iastate.edu

Stephen B. Blessing  
University of Tampa  
401 W. Kennedy Blvd.  
Tampa, FL 33606.  
813-257-3461  
sblessing@ut.edu

Keywords: Intelligent tutoring system, authoring tool, graphical user interface

ABSTRACT: We describe an evaluation of two intelligent tutoring system authoring tool paradigms, graphical user interface-based and text-based by taking the examples of CTAT and xPST respectively, in two domains, statistics and geometry. We conducted a study with 16 tutor-authors divided into 2 groups (programmers and non-programmers). Our results showed that the GUI-based approach, aided by the visualization of problem-solving strategies, provides a much lower bar for entry when compared to the text-based approach. However, the difference in tutor-authoring time between the two approaches decreases as the tutor-authors gain experience using the respective authoring tools. We also do a theoretical comparison of the two paradigms by applying the cognitive dimensions framework. This research contributes design guidance for architects of tutor authoring systems.

1. Introduction

Authoring an intelligent tutoring system from scratch is a challenging task since it requires expertise in several fields including cognitive science, computer science and pedagogy. An authoring tool tries to lower the skill threshold required for developing ITSs and also enable their rapid development (Murray, 1999). Past estimates for how long it takes to create an ITS have ranged as high as 200 hours of development time for 1 hour of instruction (Woolf & Cunningham, 1987). Recent ITS authoring tools hope to decrease that by as much as an order of magnitude.

When an authoring tool is being designed, there are several design trade-offs involved because many of the design decisions that lead to an authoring tool result from conflicting trade-offs. For example, increasing the power of an authoring tool might come at the cost of its ease of use due to the increasing complexity. In this paper, we try to evaluate the effects of the trade-offs involved in the design of GUI-based and text-based authoring tools.

Studies have been conducted to identify the advantages and disadvantages of visual programming, a common technique found in GUI-based authoring tools (Whitley, 1997). Visual representations have shown to be beneficial when the size or complexity of the problem grows (Day, 1988; Polich & Schwartz, 1974). On the other hand, visual representations do not support enough visual elements on a screen at one time, causing low screen density, when compared to textual representations. Therefore, they may not be practical for large problems (Whitley, 1997). We suggest that as tutor complexity increases, the benefits of a GUI will decrease.

For the experiment, we chose CTAT (Koedinger, Alevens, & Heffernan, 2003) and the Extensible Problem Specific Tutor, or xPST (Blessing, Gilbert, Blankenship, & Sanghvi, 2009) as examples of GUI-based and text-based authoring tools respectively. Both CTAT and xPST can be used to develop example tracing tutors (Alevens, McLaren, Sewall, & Koedinger, 2009), for both single strategy problems and multiple strategy problems. Example-tracing tutors are ones in which there is no explicit cognitive model. Therefore, the instruction contained within the tutor is applicable
to only a single problem or a small class of problems. This approach trades off wider applicability with easier authoring. However, non-programmers and non-cognitive scientists (e.g., course instructors) have used both authoring tools to create effective ITSs (Aleven, et al., 2009; Maass & Blessing, 2011).

We chose two problem domains of varying complexity from an authoring point of view—statistics and geometry. Statistics problems are sequential in nature and generally have a single solution path. Problems in geometry could have multiple strategies and therefore multiple solution paths.

1.1 Cognitive Tutor Authoring Tools (CTAT)

CTAT supports the development of example-tracing tutors through a technique called “programming by demonstration” (Koedinger, Aleven, Heffernan, McLaren, & Hockenberry, 2004; Nevill-Manning, 1993). Once the interface for a problem has been built, the tutor-author proceeds by demonstrating all possible solution paths to the problem. The interface is typically built using predefined Flash or Java widgets, with some possibility to use existing interfaces. The demonstration of the solution automatically creates a directed, acyclic graph called the “behavior graph”, which represents the acceptable ways of solving a problem (Figure 1). The feedback associated with the individual states in the problem (e.g., the hint messages displayed to the student when requested) is added to the tutor by the author through CTAT’s GUI.

Though CTAT is capable of supporting the development of both cognitive tutors and example-tracing tutors, we consider only the example-tracing tutor development feature of CTAT in this study.

1.2 Extensible Problem Specific Tutor (xPST)

xPST is an ITS authoring tool that helps rapidly develop example-tracing tutors on existing interfaces such as webpages. An example-tracing tutor is built using xPST by writing a text file that describes the sequences in which individual steps in the problem can be completed by the learner and the answers and feedback associated with each step.

Although text-based, xPST’s syntax (Figure 2) was designed such that it was simple enough for non-programmers to use, and also powerful enough for experienced users to build tutors rapidly. The code is entered using a web-based authoring tool that offers syntax-checking.

2. Methods

The experiment involved two independent variables—authoring tool paradigm (text-based or GUI-based) and problem domain (statistics or geometry). Programming level (programmer or non-programmer) was a moderating variable. Dependent variables included a rubric-based quality score for each tutor, the time to develop the tutor, and qualitative data from an exit questionnaire.

2.1 Participants

Participants were recruited through an email advertisement, fliers on campus and word-of-mouth. Sixteen participants completed the study successfully. Each participant took a pre-survey that asked questions about his or her experience with computer programming. A participant was classified as a “programmer” if he or she had taken two or more programming courses and as a “non-programmer” otherwise. The participants were divided into eight groups formed from all possible combinations of programming level (programmer or non-programmer), authoring tool paradigm (text-based or GUI-based) and problem domain (statistics or geometry), with two in each group.

2.2 Materials

The experiment used a between-subjects design: each participant was given the task of building three tutors using a specific authoring tool (CTAT or xPST) for a specific domain (statistics or geometry). All three problems were of the same complexity, as measured by their solutions having six subgoals or steps.

The entire study was conducted online. The participants were allowed to complete their tasks over multiple sessions at their own pace, over a two-week period. The participants were provided with the link to the study webpage that had all the resources and material required in order to complete their tasks. The study webpage contained a brief introduction the field of intelligent tutoring systems. The training material on the webpage included a video tutorial that gave a demo of the step-by-step procedure to be followed while creating a tutor for a sample problem as well as a text tutorial. We estimate the total training time to be about 1-2 hours, though we did not record the exact amount of time individual authors spent with the training materials.

CTAT tutor-authors used Remote Desktop Connection to log in remotely to a Microsoft Windows computer that had CTAT v2.10.0 and Adobe Flash Player 10 pre-installed. xPST tutor-authors logged into the xPST Web-based Authoring Tool (Gilbert, Devasani, Kodavali, & Blessing, 2011) using Mozilla Firefox 3 or higher. The problems for which the tutors were to be
sequence
{
  ( (stepA then stepB) or (stepC then stepD) ) then All-Done;
}

feedback
{
  stepA
  {
    answer: 60;
    Hint: “Angle A and angle B are corresponding angles.”;
  }
}

Each participant was asked to create the three tutors as if he or she was a teacher for that subject preparing homework problems for his or her students. Instructions were provided for each problem that included the problems’ solutions and the types of feedback the tutor must give for each problem, e.g., "If the student enters the median instead of the mean at step 4, display a message that says 'The mean and the median are not necessarily the same.'". This approach was used to minimize the time spent by participants on pedagogical design. The tutors created by the participant were meant to monitor each step in the corresponding problem. For each problem step, the tutor was to provide exactly one hint. Also, for each tutor overall, there was to be one message that gave

2.3 Procedures

built were predefined and the interfaces for the problems were provided to the participants.

CTAT tutor-authors could access the problem interfaces (.swf files) on the remote machine provided to them. xPST tutor-authors were provided with the links to the webpages that contained the problem interfaces. We conducted the study in this way to ensure we were truly investigating just the time to use the ITS authoring system and not the time to set up the system (which was more time consuming for CTAT than xPST) nor the time to set up the problem interface, which is more of an interface or webpage design task.
feedback for a specific error that a student might make, e.g. the message mentioned above.

After successful completion of their tasks, the participants received a compensation of $40 in cash and a chance to win $149 in cash through a lottery.

3. Results

We had a total of eight groups with two participants each. Each participant built a total of three tutors, leading to the creation of 48 tutors in all.

3.1 Model Analysis

Each tutor was scored on two criteria – “Solution Path” and “Error-Specific Feedback.” A tutor was given a score of 1 under “Solution Path” if it correctly provided tutoring for all possible solution paths in the problem (including providing hints on each step), 0.5 if it correctly provided tutoring for one of the possible solution paths and 0 if it provided completely incorrect tutoring. A tutor received a score of 1 under “Error-Specific Feedback” if it correctly provided the required error-specific feedback and 0, otherwise.

The cumulative scores have been shown in Table 1. Since a group had two participants who built three tutors each, the maximum score possible is six. The model analysis shows that all the tutor-authors who were classified as programmers built tutors that provided accurate tutoring. Tutor-authors who were classified as non-programmers built tutors that displayed accurate tutoring behavior for statistics problems, but slightly less accurate behavior for geometry problems, which were the problems that allowed for multiple solution paths.

3.2 Timing Data

The total time spent in creating each tutor was logged by the respective authoring tools. The web-based authoring tool used by xPST participants calculated the time spent developing a tutor as the sum of the time spent in editing the code and the time spent testing it. The CTAT logger creates a log file that records the time and date, each time the tutor-author interacts with the GUI, from which the total time developing a tutor was calculated. This total time measure for both CTAT and xPST includes the time spent in authoring the tutor as well as testing it.

Figure 3 shows the histogram of the time spent in minutes by the xPST tutor-authors, in building tutors for all three problems.

Figure 4 shows the histogram of the time spent in minutes by the CTAT tutor-authors, in building tutors for all three problems. The log file for the second tutor created by one of the participants (P11) was unavailable.

Figure 5 shows the learning curve for the tutor-authors. It is interesting to see that after the tutor-authors gained experience building three tutors, the average time required in creating a tutor by the groups xPST–Statistics (19 min), CTAT–Statistics (18.75 min) and CTAT–Geometry (18 min) were almost equal. However, the average time required in creating a tutor for the third geometry problem using xPST (52 min) was much higher than the average time required using CTAT (18 min). The geometry problems that we chose involved multiple solution strategies. The results suggest that subtle ordering of steps in multiple-strategy problems is more convenient in CTAT because of CTAT’s visual representation of the strategies on the behavior graph.

3.3 Exit Questionnaire Data

All 16 participants answered a short questionnaire after completing their tasks. The questionnaire asked them for their feedback about the authoring tool they had used to create tutors. They were asked to rate the ease of use and power of the authoring tool on a Likert scale of 1 to 5, and to answer open-ended questions about the tool’s strengths, weaknesses, and suggestions for improvement. One common theme that emerged from the open-ended questions was that both xPST and CTAT are easy to author once understanding how they

<table>
<thead>
<tr>
<th>Authoring Tool</th>
<th>Problem Domain</th>
<th>Programmer / Non-programmer</th>
<th>Solution Path Score</th>
<th>Error-Specific Feedback Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>xPST</td>
<td>Statistics</td>
<td>Programmer</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-Programmer</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Geometry</td>
<td>Programmer</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-Programmer</td>
<td>4.5</td>
<td>5</td>
</tr>
<tr>
<td>CTAT</td>
<td>Statistics</td>
<td>Programmer</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-Programmer</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Geometry</td>
<td>Programmer</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-Programmer</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 1: Cumulative scores of created tutors (maximum possible score is 6)
work. Some quotations are included below that illustrate these:

“Very easy to use once you get a feel for the syntax.” – P7 (xPST – Non-Programmer)

“Once we [sic] understand how to create the tutor then the tool is very simple to use.” – P14 (CTAT – Programmer)

The average ratings from the exit questionnaire have been summarized in Table 2. Both xPST and CTAT were rated slightly higher by programmers for their ease of use when compared to non-programmers.

4. Discussion and Future Work

We used a between-subjects experimental design to prevent order effects and contamination as the participants moved between authoring systems and
domains. We felt that the comparisons between levels of the independent variables would be compromised if the variables were to be within-subject. As a result, we cannot claim that our results are statistically significant, partially because of the small sample size.

4.1 Applying the Cognitive Dimensions Framework

While the experimental data from this pilot study offers information about learning curves, it is also useful to apply elements of Green & Petre's cognitive dimensions framework (1996), which is helpful for usability analysis generally and was originally used to compare visual programming languages with BASIC. While the framework contains 13 cognitive dimensions, we choose a subset of those for the most applicable comparison.

**Closeness of mapping** is the degree to which elements and operations within the programming language can be mapped directly onto objects and actions within the problem domain. A more direct mapping is preferred. Typically, a visual programming language like CTAT has a closer mapping than a textual language like xPST, especially since textual languages have syntax constraints that do not relate to the problem domain at all. However, xPST has been designed with close mapping in mind, in that steps through the problem correspond to components of code. Mappings in visual programming languages can suffer when looping or branching needs to occur based on calculations of variables, but in the simpler geometry and statistics problems of this study, that issue does not arise. For closeness of mapping, CTAT likely has some advantages over xPST.

**Diffuseness/Terseness** relates to the number of symbols or lexemes needed to express a concept, in this case, a tutor's structure, and is related to the amount of representation that is viewable simultaneously on screen. In this case, visual programming languages typically suffer once the program reaches a reasonable degree of complexity. In the simpler tutor problems we used, screen real estate was not an issue, but we suggest that with more branches or complex paths to a goal, CTAT would be more diffuse than xPST.

**Premature Commitment** refers to the extent to which the authoring tool forces the user to make decisions before complete information is available, which often happens when components have significant interdependencies and there are no order constraints. In text-based languages, programmers typically insert stubs to address this issue and return to complete them later. This is the case with xPST. Visual programming languages often force a developer to commit to a certain layout of objects and a certain set of connections before the final layout and connections are finalized. If it is difficult to change these after the system is built, the language has high premature commitment. While CTAT's layout and connections between states are relatively easy to adjust, with higher complexity, xPST likely has some advantages in this dimension.

**Secondary Notation** refers to the tool's ability to convey meaning about the structure of the program with methods beyond the elements of the program itself. In xPST and other text-based tools, for example, indentation, commenting, and blank lines help illustrate the coherent chunks of the tutor. Some visual programming languages allow boxes or regions to highlight a grouping of related elements, but not all, and often developers with these visual tools spend considerable time not doing the "programming" itself but simply rearranging elements to group or align them appropriately. CTAT supports naming of states and grouping of related states, which aids in visualizing the solution paths. Also, it uses different colors to differentiate between correct and incorrect actions in the behavior graph.

**Error-proneness** refers to the extent to which the authoring tool induces careless mistakes from the author. The syntax of textual programming languages might be more error-prone, compared to visual programming languages. Moreover, errors might be harder to detect in textual programming languages, especially if the errors are not syntactic in nature. Though xPST was designed such that its syntax is simple to use, it still requires a semicolon as a separator, which could cause errors by non-programmers. Since CTAT uses a graphical user interface to develop a tutor, such careless mistakes occur less often.

**Visibility and Juxtaposibility** refer to the number of steps required to make desired information visible (the former) and the ability to see separate portions of the system at the same time (the latter). Systems in which visual components hide nested scripts or calculations typically have low visibility and juxtaposibility, requiring the user to right-click to reveal information one component at a time. While CTAT graph nodes contain several pieces of information in their default display, the user is required to dig in to edit information within a node (lower visibility). It is difficult to see the contents of two nodes simultaneously (lower juxtaposibility). Text-based systems like xPST typically have higher visibility (because all the code is present) and high juxtaposibility (via multiple windows).

While we have not addressed all of the cognitive dimensions, we have touched on those that are likely of...
greatest interest in the context of comparing xPST and CTAT. According to this theory, each tool has some advantages.

4.2 Choose the Best of Both Tools

Generally, a graphical user interface-based approach allows for easier learning initially and a lower bar for entry, as confirmed by the collected data. However, as noted by the theory of cognitive dimensions, once a complex tutor is built, it can be time consuming to edit because of the need to change parameters in multiple locations within the visual structure that are hidden to ensure good visual presentation. One of the advantages of the text-based approach is that debugging and editing an existing tutor may be easier since the entire code is available to the tutor-author at one glance. We conclude by proposing that the best way forward for authoring intelligent tutoring systems and behavior models is a hybrid authoring tool that exploits the synergy between the graphical user interface-based paradigm and the text-based paradigm. Much like common integrated development environments (IDEs) such as Adobe Dreamweaver and Microsoft Visual Studio, the ideal tutor authoring tool would have a “design” tab, which affords tutor-creation through a GUI and a “source” tab that supports the editing of the tutor directly through code. We expect such a tool would cater to programmers and non-programmers, experienced and non-experienced.

5. References


Author Biographies

SHRENIK DEVASANI recently graduated with his MS in Human Computer Interaction and Computer Science from Iowa State University and is now a research assistant at the Palo Alto Research Center’s Intelligent Systems Laboratory. His work involves the building of tools in support of work-process optimization.

STEPHEN GILBERT is Associate Director of the Virtual Reality Applications Center at Iowa State University. His background includes cognitive science and industrial engineering, and he helps direct Iowa State's graduate program in Human Computer Interaction. He is PI on a Live, Virtual and Constructive training contract for the U.S. Army RDECOM STTC.

STEPHEN BLESSING is an Associate Professor of Psychology at the University of Tampa. His main research interest is in authoring tools for intelligent tutoring systems, and he has co-edited a book on the topic.