A Model for Deploying Software Tutors

Amruth N. Kumar
Computer Science
Ramapo College of New Jersey
Mahwah, NJ 07430, USA
amruth@ramapo.edu

Abstract—We propose a model for deployment of software tutors. In the model, the first stage of the pre-test-practice-post-test evaluation protocol is used to initialize the student model needed for adaptation. In order to keep assessment congruent with learning, the same type of activity is used in all three stages of evaluation, and the software tutor itself is used to administer all three stages of the protocol. In order to minimize time while maximizing learning, post-test is set to adapt to practice, and students are also provided feedback after pre-test activities on which they provide incorrect response. Finally, the software tutor is set up to administer the pre-test-practice-post-test protocol as one seamless integrated session with no discernible breaks between the stages. This model of deployment helps the developer evaluate the effectiveness of the software tutor, helps the student minimize time while maximizing learning and enables the instructor to obtain concept-level summary of what worked and what needs to be reviewed again in the classroom. The deployment model has been implemented in software tutors for programming, called problets (problets.org). We produce quantitative data from three problets used in spring 2013 to illustrate the benefits of the deployment model that accrue to developers, students and instructors.

Keywords-component; Software tutors, Evaluation, Adaptation, Deployment

I. INTRODUCTION

Software tutors are increasingly being used over the web to help students learn targeted Science, Technology, Engineering and Mathematics (STEM) topics through activities such as problem-solving, visualization, and simulation. These tutors have the advantage that they facilitate self-paced learning, active learning [1] and learning anytime, anywhere.

Multiple actors are involved in the development and use of software tutors: 1) Developers, typically educators or researchers, who develop the software tutors and make them available for public use; 2) Instructors who select the tutors for use in their courses and recommend or require their students to use them; and 3) Students or learners who use the software tutors, either in class/closed lab or after class as an assignment.

The needs and goals of these three actors are not always the same. Developers are keen on learning whether the software tutors are effective at helping students learn. This may be born of a practical interest in the utility of the tutors they developed, or may feed into a larger research agenda. The instructors who adopt the software tutors would want to know whether there are particular concepts that everyone in their class finds hard; whether the adopters should dedicate additional coverage in class to such concepts. The students themselves, usually, want to spend as little time as possible working with the software tutors – at least no more time than is necessary to get course credit for doing so.

Given these differing needs and goals, we propose a model of deployment of software tutors that will try to satisfy all three actors – students spend no more time than is necessary, instructors can get detailed concept-level reports of what was not covered well in class and what needs to be covered again; and developers get quantitative data on the effectiveness of their software tutors.

II. A DEPLOYMENT MODEL FOR SOFTWARE TUTORS

A. Merging Adaptation and Evaluation

Learning is most effective when it is adapted to the needs of the learner [2]. In a tutor, various aspects can be adapted to the needs of the learner, including the problem sequence, feedback type, and the feedback amount. Vector spaces [3] and learning spaces [4] are the popularly used mechanisms for adaptation of problem sequence in tutors. We had earlier proposed associative adaptation [5] as a scalable domain-independent alternative. In associative adaptation, student model is an overlay of the domain model which is constructed as a hierarchy of domain concepts; problems are associated with specific domain concepts; and adaptation occurs in terms of the domain concepts – students are presented problems on the domain concepts on which mastery has not been demonstrated in the student model. Personal human tutors are estimated to increase performance by as much as two standard deviations [2]. The goal of software tutors is to achieve at least half that improvement, i.e., one standard deviation. In order to do so, they must adapt to the needs of the learner, making adaptation a required part of any software tutor. However, in order for adaptation to be successful, the student model must first be initialized, i.e., the current level of knowledge of the student must be assessed. Earlier researchers have proposed using a test to initialize the student model needed for adaptation [6, 7]. Other alternatives proposed
for initializing the student model include stereotypes [8] and schema-based assessment [9].

Pre-test-practice-post-test is the classical protocol used to evaluate treatments. Typically, pre-test and post-test are matched in levels of difficulty. They are designed to assess the learner’s level of knowledge on specific topics and concepts. The treatment is provided during practice: its effect on the learner’s level of knowledge is measured by the change in score from pre-test to post-test. A pre-post increase signals positive effect of the treatment on learning. When software tutors are evaluated, they are administered as the treatment during practice.

We propose that pre-test, the first stage in the pre-test-practice-post-test protocol be used to initialize the student model needed for adaptation. This approach has multiple benefits: through adaptation, it minimizes the time spent by the student using the tutor, while maximizing learning. At the same time, by employing pre-test-practice-post-test protocol, it enables the developer to obtain quantitative information about the effectiveness of the software tutor.

In this deployment model, students first answer a pre-test, consisting of one activity per domain concept. Their student model is initialized based on their performance during the pre-test. Next, the software tutor presents activities to the students during the practice stage. It adapts to the outcome of the pre-test – it presents activities on only those domain concepts on which the student provided incorrect response during pre-test. Finally, the student is asked to answer a post-test as per the protocol.

B. Administering the Entire Evaluation Protocol

Software tutors typically step the student through an activity intrinsically designed to help the student learn domain concepts. Typical activities include problem-solving, animation and visualization in Computer Science. If these activities are proven to help students learn, by their intrinsic nature, they are also effective at assessing the student’s learning. So, it is logical to use the same type of activity for pre-test, practice and post-test, e.g., students solve problems during all three stages, as opposed to solving problems during practice, but answering multiple-choice questions during pre-test and post-test. The advantage of using the same activity through all three stages is that there is no mismatch between learning and assessment.

If the same type of activity is used for pre-test, practice and post-test, we propose that the software tutor itself administer all three stages of the evaluation protocol. Typically, what makes a software tutor effective is not the activity per se, but the feedback provided by the tutor when a student fails at an activity. By configuring the tutor not to provide such feedback, it can be used to administer pre-test and post-test also. Having the software tutor administer all three stages has the added advantage that students do not need to switch contexts or deal with disparate interfaces/media.

Typically, a software tutor provides two types of feedback when a student makes a mistake: error-detection and error-correction. The feedback provided by intelligent tutoring systems has been categorized into three types [10]:

- Delayed feedback, when both error-detection and error-correction feedback are provided after the student submits his/her response to the activity;
- Error-flagging, when error-detection support is provided while the student is attempting the activity, but error-correction support is presented only after the student has submitted his/her response;
- Immediate feedback, when both error-detection and error-correction support are provided while the student is attempting the answer.

The deployment model where the software tutor administers pre-test and post-test by withholding feedback works for both delayed feedback and error-flagging, but not immediate feedback.

C. Minimizing Time, Maximizing Learning

Since practice is adaptive, we propose that post-test should be adaptive too. Since students carry out activities during practice on only those concepts on which they failed during pre-test, it makes sense to assess their learning on only these domain concepts during post-test. This has the advantage of reducing the number of post-test problems solved by students while providing a more accurate picture of the learning of the students.

Typically, no feedback is provided after the pre-test activities. However, if a student fails an activity during pre-test, additional learning can be facilitated by providing feedback for the activity, especially if error-correction support is provided only after the student submits his/her response, i.e., delayed feedback or error-flagging feedback is provided. With the provision of feedback, the pre-test activity doubles as a practice activity, giving the student additional opportunity to learn, and potentially reducing the number of activities the student must complete with the software tutor.

D. Integrated Session

Software tutors may be used in class, in closed labs or after class as assignments. In higher education, given the demands on classroom time, software tutors are usually assigned for use after class. Given this, evaluation of software tutors in higher education is usually carried out in-natura. In-natura evaluation differs from other types of evaluation in the following respects:

- In in-ovo evaluation, research subjects picked for the evaluation use the software tutors in a laboratory setting, typically under tightly controlled conditions, and under supervision.
- In in-vivo evaluation, actual students use the software tutors in the class room, typically under tightly controlled conditions and under supervision.
- In in-natura evaluation, actual students use the software tutors, typically after class, on their own time, and unsupervised.

Since students use software tutors unsupervised and on their own time, their motivation may be not so much to learn from the software tutor as to get done with the
assignment in as little time as possible while still getting credit for completing it. This complicates the administration of pre-test-practice-post-test protocol: in our experience, students tend to drop out of the session before completing all three stages. This results in unusable evaluation data, not to mention, an incomplete learning experience for the student.

To counter this, we propose that the software tutor administer pre-test-practice-post-test protocol as one seamless session with no discernible breaks between the stages. In such a seamless session, students are presented pre-test, adaptive practice and adaptive post-test activities interspersed, under the following conditions:

- A student is never presented practice activity on a domain concept until after the pre-test activity. If a practice activity involves more than one domain concept, it is not presented until after the pre-test activity for all its domain concepts.
- A student is never presented post-test activity on a domain concept until after sufficient number of practice activities are presented for the student to master the domain concept.

In order to prevent predictability of domain concepts underlying activities, and ensure authenticity of any learning that occurs, the activity scheduler might avoid certain sequences of activities such as back-to-back pre-test, practice and post-test activities for the same domain concept. It may use any of a number of scheduling algorithms such as round-robin or least-known-first for scheduling practice activities on domain concepts. It may use eager scheduling algorithms for post-test activities to maximize the number of domain concepts on which the student carries out activities in all three stages, especially if the entire session is time-limited.

E. Analysis by Domain Concepts

There are many reasons for limiting the duration of use of a software tutor – the attention span of the learner should not outlast the session with the software tutor; and in-class use of software tutors would be constrained by the time an instructor would want to be able to allocate to using the tutor. When a time limit is imposed, an integrated software tutor, i.e., a software tutor that administers pre-test, practice and post-test in one seamless session may run out of time before presenting all three stages on all the domain concepts. In other words, the learner may complete only one, two or all three stages on the different domain concepts. We classify domain concepts as follows in an integrated software tutor:

- **Known:** Student provides the correct response on the pre-test activity. So, the student knows the domain concept and need not carry out any practice or post-test activities on the concept.
- **Out of time:** The student provides incorrect response on pre-test activity. The student runs out of time either before or during practice activities, or before the post-test activity. Unfortunately, these domain concepts cannot be included in any evaluation of the software tutor or the learner.
- **Practiced:** The student provides incorrect response on pre-test activity, engages in sufficient practice activities to master the domain concept, and completes the post-test activity on the concept. If the student demonstrates improvement in pre-post score, the concept is also **learned**.

All three types of domain concepts might result from a single student session with an integrated software tutor - some concepts may be known, some practiced, and the rest out of time. So, analysis of the results of evaluation will have to be carried out not in terms of activities (e.g., problems solved), but rather, in terms of domain concepts. The report provided for each student would consist of three types of domain concepts:

- Knew before using the software tutor – these are the known concepts
- Learned after using the software tutor – these are the learned concepts
- May still need to learn: these are all out of time concepts and any practiced concepts that have not yet been learned.

An advantage of analyzing the data collected by integrated software tutors in terms of domain concepts is the ability to provide a report to the instructor on what worked and did not work during classroom instruction:

- Domain concepts that were **covered well** in class are those with high class average scores on the pre-test activity.
- Vice versa, domain concepts that were **not well understood** during classroom instruction are those with low class average scores on the pre-test activity.
- Domain concepts that may have to be **reviewed again** in class are those with lower class average scores on pre-test activity that most students may still need to learn because they ran out of time.

The primary purpose of the traditional report consisting of the number of activities attempted by each student and the score of the student on those activities is assessment – how much course credit to award to the student for working with the software tutor. On the other hand, the primary purpose of a report of domain concepts as described above (covered well, not well understood, to be reviewed again) is to improve classroom instruction.

In summary, we proposed a model for deployment of software tutors. In the model, the first stage in the pre-test-practice-post-test evaluation protocol is used to initialize the student model needed for adaptation. Given the need for congruence between learning and assessment, the type of the activity is kept the same among the three stages of evaluation, and the software tutor itself is used to administer all three stages of the protocol. In order to minimize time while maximizing learning, post-test is also set up to be adaptive, and students are also provided feedback after pre-test activities on which they provide incorrect response. Finally, the software tutor is set up to administer the pre-test-practice-post-test protocol as one seamless integrated session with no discernible breaks between the stages.
This model of deployment helps the developer evaluate the effectiveness of the software tutor, helps the student minimize time while maximizing learning and enables the instructor to obtain concept-level summary of what worked and what needs to be reviewed again in the classroom.

III. EVALUATION

Problets (problets.org) are software tutors developed to help introductory programming students learn programming constructs by solving problems. Problets are meant to be used as supplements to classroom instruction and complements (rather than substitutes) to traditional programming projects. Each problet deals with one topic, such as arithmetic expressions, while loops, one-dimensional arrays or C++ pointers. Within each topic, a problet targets 9-25 domain concepts, such as nested loops, parameter passing by value, correct evaluation of modulus operator, etc. To date, 17 problets have been developed to cover all the topics typically included in an introductory programming course.

Problets have been continually used by third parties over the web since fall 2004. During that time, problets started out administering distinct pre-test, practice and post-test stages, but switched to integrated model in fall 2007. Since then, problets have been used by over 300 instructors/institutions, mostly in the United States, but some also in Europe and Asia. So, we have gathered a wealth of experience and data using the proposed deployment model for problets over the last 7 years (e.g., [11]).

Some of the features of problets are that they provide step-by-step explanation of the correct answer after the student has submitted his/her incorrect answer, which has been shown to improve learning [12], they provide error-flagging feedback while the student is attempting his/her answer [13], and they adapt to the needs of the learner [5].

In the following subsections, we illustrate the advantages of the proposed deployment model for developers, students and instructors by citing actual data from evaluation of three problets in spring 2013, on arithmetic expressions, for loops and debugging functions.

A. Arithmetic Expression Tutor

Arithmetic expression tutor covers 5 arithmetic operators, their correct evaluation, precedence, associativity, coercion and errors. In the tutor, students are asked to evaluate arithmetic expressions one operator at a time. In spring 2013, users of the tutor were divided into two groups: one group was provided feedback with visualization while the other was provided feedback without. The visualization involved using underbraces to display the steps in the evaluation, as shown in Figure 1. Underbraces duplicated the text explanation also provided as part of the feedback.

For developers: One advantage of the proposed model of deployment is that the software tutor is evaluated every time a student uses it. If a student practices even one concept, the data can be used to evaluate the effectiveness of the tutor.

<table>
<thead>
<tr>
<th>Arithmetic Tutor</th>
<th>No Viz</th>
<th>Viz</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>77</td>
<td>69</td>
</tr>
<tr>
<td>Concepts</td>
<td>1.429</td>
<td>1.507</td>
</tr>
<tr>
<td>Pre-test Mean Score</td>
<td>0.032</td>
<td>0.017</td>
</tr>
<tr>
<td>Practice Problems</td>
<td>5.026</td>
<td>5.275</td>
</tr>
<tr>
<td>Post-test Mean Score</td>
<td>0.927</td>
<td>0.948</td>
</tr>
<tr>
<td>Significance</td>
<td>$p &lt; 0.001$</td>
<td>$p &lt; 0.001$</td>
</tr>
</tbody>
</table>

Table 1 shows the number of subjects, concepts learned, pre-test mean score, number of practice problems solved, post-test mean score and 2-tailed t-test $p$ value for the two groups that used Arithmetic tutor. It is clear from the table that whether visualization was provided or not, the tutor helped students learn concepts.

For students: On arithmetic tutor, 58 students in the no-visualization condition and 85 students in the visualization condition solved all 16 pre-test problems and scored 100% on the pre-test. These students therefore did not have to solve any practice or post-test problems. Even though students were allowed 30 minutes to complete the tutoring session, the students in the no-visualization condition spent only 12.57 minutes with the tutor whereas those in the visualization condition spent only 14.26 minutes.

For instructors: Each problet generates a score for each student and domain concept, which is calculated as the total score on the problems involving the concept, divided by the number of problems solved involving the concept. The problems considered for this cumulative score are from all three stages of evaluation: pre-test, practice and post-test. The lower the average score of students for a domain concept, the more the students who solved problems incorrectly on that domain concept. The concepts on which students in both the groups solved the most problems incorrectly were:

- Coercion of operands in a mixed-mode expression involving multiplication
- Application of modulus operator to real operands

The group with no visualization condition also found correctly evaluating integer division expression to be hard. The group with visualization treatment also found correctly evaluating the modulus operator (with integer operands) to be hard. Further evaluation is needed to clarify when visualization works and why.

B. Counter-Controlled Loop Tutor

Counter-controlled (for) loop tutor covers concepts such as zero-iteration loops, up-counting loops, down-counting loops, nested loops and back-to-back loops. In the tutor, students are asked to identify the output of a program containing a loop, one output at a time. The tutor provided error-flagging feedback, i.e., error-detection, but not error-correction support while the student was attempting the problem. In spring 2013, users of the tutor
were divided into two groups: those who could revise their answer as many times as they pleased, and the other group who could revise their answer at most three times per problem.

**For developers:** Table II shows the number of subjects, concepts learned, pre-test mean score, number of practice problems solved, post-test mean score and 2-tailed t-test $p$ value for the two groups that used for loop tutor. It is clear from the table that whether the number of revisions was limited or not, the tutor helped students learn concepts.

### Table II. For Loop Tutor: Spring 2013

<table>
<thead>
<tr>
<th></th>
<th>for Loop Tutor</th>
<th>No Limit</th>
<th>Limit of 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>69</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>Concepts</td>
<td>1.304</td>
<td>1.448</td>
<td></td>
</tr>
<tr>
<td>Pre-test Mean Score</td>
<td>0.311</td>
<td>0.266</td>
<td></td>
</tr>
<tr>
<td>Practice Problems</td>
<td>3.855</td>
<td>4.648</td>
<td></td>
</tr>
<tr>
<td>Post-test Mean Score</td>
<td>0.983</td>
<td>0.960</td>
<td></td>
</tr>
<tr>
<td>Significance</td>
<td>$p &lt; 0.001$</td>
<td>$p &lt; 0.001$</td>
<td></td>
</tr>
</tbody>
</table>

**For students:** On for loop tutor, 12 students in the no-limit condition and 16 students in the limited revisions condition solved all 10 pre-test problems and scored 100% on the pre-test. Even though they were allowed 30 minutes to complete the tutoring session, students in the no-limit condition spent only 15.48 minutes with the tutor whereas those in the limited revision condition spent only 12.61 minutes.

**For instructors:** The concepts on which students in both groups solved the most problems incorrectly were:
- Nested loops, whether or not the nested loop counter depended on the nesting loop’s counter
- Back-to-back loops, where the second loop’s counter depended on the terminal value of the first loop’s counter
- Loop with a simple statement as its body

**C. Debugging Functions Tutor**

The tutor on debugging functions addresses common errors in function definition and call such as mismatch in the number and data type of formal and actual parameters, missing return statement in the definition of a function and incompatibility of the return type and return value of a function. The students are asked to identify these errors. In spring 2013, the users of the tutor were divided into two groups – those who were asked to answer a questionnaire on user interface issues upfront versus those who were asked to answer a questionnaire addressing their anxiety about using the software tutor.

**For developers:** Table III shows the number of subjects, concepts learned, pre-test mean score, number of practice problems solved, post-test mean score and 2-tailed t-test $p$ value for the two groups that used the tutor. It is clear from the table that the tutor helped students in both groups learn concepts.

**For students:** Only one student solved all 8 pre-test problems correctly. The session for that student lasted only 14.58 minutes, although students were allowed 30 minutes to complete the tutoring session.

### Table III. Debugging Functions Tutor: Spring 2013

<table>
<thead>
<tr>
<th>Debugging Functions</th>
<th>User Interface</th>
<th>Anxiety</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>84</td>
<td>14</td>
</tr>
<tr>
<td>Concepts</td>
<td>2.607</td>
<td>3.50</td>
</tr>
<tr>
<td>Pre-test Mean Score</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Practice Problems</td>
<td>9.024</td>
<td>13.643</td>
</tr>
<tr>
<td>Post-test Mean Score</td>
<td>0.882</td>
<td>0.962</td>
</tr>
<tr>
<td>Significance</td>
<td>$p &lt; 0.001$</td>
<td>$p &lt; 0.001$</td>
</tr>
</tbody>
</table>

**For instructors:** The concepts on which students in both groups solved the most problems incorrectly include:
- Return statement missing in the definition of a function with non-void return type
- Formal parameter is re-declared as a local variable in the definition of a function

Concepts on which one group or the other solved the most problems incorrectly include:
- Mismatch in the number of actual and formal parameters
- Value returned by the function is ignored by the caller
- Function is called before it is defined/prototyped ($C++$)

This is the type of information that helps instructors review or revise coverage in their course.

**D. Discussion of Results**

As is clear from the results of the three problems, students learned more using tutors on advanced concepts (e.g., debugging functions) than on basic concepts (e.g., arithmetic expressions), and this was a function of necessity, as shown by the fewer students who scored 100% on the pre-test on advanced concepts.

While the examples and the benefits presented earlier were specific to introductory programming concepts, the model of deployment can be used for any discipline that involves problem-solving, STEM or otherwise. As long as problems are explicitly associated with domain concepts and these domain concepts are used for adaptation, instructors can obtain concept-level summary of what worked and what needs to be reviewed again in the classroom. By virtue of availing adaptation, students can minimize the time spent while maximizing learning. Because of the built-in pre-test-practice-post-test protocol, developers can evaluate the software tutor every time a student uses it. The more the users of the software, the more the evaluation data collected, making this model of deployment scalable and especially suitable for use in large classes and MOOCs.

Software tutors are typically deployed to administer problem-solving practice session only, e.g., CodeLab (TuringsCraft.com). They are not also designed to administer evaluation protocol. When software tutors are evaluated, especially in Computer Science, the evaluation is typically carried out in-ovo (e.g., [14]). Such evaluation
typically involves pre-test, practice and post-test as three distinct stages. This means, whether the student learned from using the software tutor or not, the student still goes through the entire protocol. Moreover, the instruments used for pre-test and post-test engage in a different activity (e.g., filling out a survey as in [14]) or medium (e.g., [15]) than the problem-solving software tutor itself, leading to a mismatch between assessment done during pre-test and post-test and learning affected during the intervening practice. The model of deployment presented in this paper tries to address these shortcomings of current practices for evaluating software tutors.

In the proposed model, the time allocated for using a software tutor must be balanced against the level of difficulty of the concepts involved. In other words, the time duration for an integrated session should be set so that the number of out-of-time concepts is minimized for the average student. If increasing the time allowed is not an option, as is frequently the case in in-natura evaluation, reducing the number of domain concepts covered by the tutor might be the solution. Another approach for reducing the number of out-of-time concepts might be to use constraint-programming to better schedule pre-test, practice and post-test activities within an integrated session. We plan to explore these options in the future.

ACKNOWLEDGMENT

Partial support for this work was provided by the National Science Foundation under DUE-0817187 and DUE-1432190.

REFERENCES

Figure 1. Screen shot of Arithmetic Expression Evaluation tutor – Problem and student attempt in the left panel and feedback in the right panel.