8-1-2008

The ASSISTment Builder: Supporting the Life Cycle of ITS Content Creation

Leena Razzaq
Worcester Polytechnic Institute, leenar@wpi.edu

Jozsef Patvareczki
Worcester Polytechnic Institute, patvarcz@wpi.edu

Shane F. Almeida
Worcester Polytechnic Institute, almeida@wpi.edu

Manasi Vartak
Worcester Polytechnic Institute, mvartak@wpi.edu

Mingyu Feng
Worcester Polytechnic Institute, mfeng@wpi.edu

See next page for additional authors

WPI-CS-TR-2008-06 August 2008

The ASSISTment Builder: Supporting the Life Cycle of ITS Content Creation

by

Leena Razzaq
Jozsef Patvarczki
Shane F. Almeida
Manasi Vartak
Mingyu Feng
Neil T. Heffernan
Kenneth R. Koedinger

Computer Science Technical Report Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department

100 Institute Road, Worcester, Massachusetts 01609-2280

The ASSISTment Builder: Supporting the Life Cycle of ITS Content Creation

Leena Razzaq[1], Jozsef Patvarczki[1], Shane F. Almeida[1], Manasi Vartak[1], Mingyu Feng[1], Neil T. Heffernan[1] and Kenneth R. Koedinger[2]
[1]Worcester Polytechnic Institute
Department of Computer Science
100 Institute Road
Worcester, MA 01609
{leenar, patvarcz, almeida, mvartak, mfeng, nth}@wpi.edu
www.assistment.org
[2]Carnegie Mellon University
Human Computer Interaction Institute
5000 Forbes Avenue
Pittsburgh, PA 15213
koedinger@cmu.edu

Abstract
Content creation is a large component of the cost of creating educational software. For intelligent tutoring systems, estimates are that approximately 200 hours are required for every hour of instruction. We present an authoring tool designed to reduce this cost. The ASSISTment Builder is a tool that is designed to effectively create, edit, test, and deploy pseudo-tutor content. The web-based interface simplifies the process of tutor construction to allow users with little or no programming experience to develop content. Previously, we have shown the effectiveness of our Builder at reducing costs to 30 hours for every hour of instruction. In this paper, we replicate this experiment and report our new results for the cost. We also describe new features that work towards supporting the life cycle of ITS content creation through maintaining and improving content as it is being used by students.

Keywords: ITS, content creation, collaboration, content builder

INTRODUCTION

Although intelligent tutors have been shown to produce significant learning gains in students [1], [8], few intelligent tutoring systems (ITS) have become commercially successful, such as Carnegie Learning's Cognitive Algebra Tutor [2]. The high cost of building intelligent tutors may contribute to their scarcity and a significant part of that cost concerns content creation. Murray [12] asked why there are not more ITS and proposed that a major part of the problem was that there were few useful tools to support ITS creation. In 2003, Murray, Blessing, and Ainsworth [13] reviewed 28 authoring systems for learning technologies. Unfortunately, they found that there are very few authoring systems that are of "release quality", let alone commercially available. Two systems that seem to have "left the lab" statge of development are worth mentioning: APSPIRE [10] an authroing tool for Contraint Based Tutors [11] and Carnegie Learing researchers [3] presented their work on creating an authoring tool for cogntive tutors. Since the focus is on building cognitve tutors their tool focuses on creating a GUI for writing

production rules. Writing production rules is naturally a difficult software engineering task, as flow of control is hard to follow in production systems.

Murray, after looking at many authoring tools [12] said, "A very rough estimate of 300 hours of development time per hour of on-line instruction is commonly used for the development time of traditional CAI." While building intelligent tutors systems is generally agreed to be much harder, Anderson [2] suggested it took at least 200:1 hours to build the Cognitive Tutor.

We hope to lower the skills needed to author ITS content to the point that normal classroom teachers can author their own content. Our approach is to allow users to create pseudo-tutors [7] via the web to reduce the amount of expertise and time it takes to create an intelligent tutor, thus reducing the cost. The goal is to allow both educators and researchers to create tutors without even basic knowledge of how to program a computer. Towards this end, we have developed the ASSISTment System; a web-based authoring, tutoring, and reporting system.

Worcester Polytechnic Institute (WPI) and Carnegie Mellon University (CMU) were funded by the Office of Naval Research (which funded much of the CMU effort to build Cognitive Tutors) to explore ways to reduce the cost associated with creating cognitive model-based tutors used in ITS [7]. In the past, ITS content has been authored by programmers who need PhD-level experience in AI computer programming as well as a background in cognitive psychology. The attempt to build tools that open the door to non-programmers led to Cognitive Tutor Authoring Tools (CTAT) [1] which two of the authors of this paper had a hand in creating. ASSISTments emerged from CTAT and shares some common features, with ASSISTments' main advantage of being completely web-based.

In this paper, we describe the ASSISTment Builder which is used to author math tutoring content and we present our estimate of content development time per hour of instruction time. With our server based system, we are attempting to support the whole lifecycle of content creation that includes error correction and debugging as well. We also describe our efforts to incorporate variablization into the Builder. Finally, we present our work towards easing the maintenance, debugging and refining of content.

I. THE ASSISTment SYSTEM

The ASSISTment project is joint research conducted by Worcester Polytechnic Institute and Carnegie Mellon University and is funded by grants from the Department of Education, the National Science Foundation, and the Office of Naval Research. The ASSISTment project's goal is to provide cognitive-based assessment of students while providing tutoring content to students.

The ASSISTment system aims to assist students in learning the different skills needed for the Massachusetts Comprehensive Assessment System (MCAS) test or (other state tests) while at the same time assessing student knowledge to provide teachers with fine-grained assessment of their students' knowledge; it assists while it assesses. The system assists students in learning different skills through the use of scaffolding questions, hints, and incorrect messages (or buggy messages) [17]. Assessment of student performance is provided to teachers through real-time reports based on statistical analysis. Using the web-based ASSISTment system is free and only requires registration on our website; no software need be installed. Our system is primarily used by middle- and high-school teachers throughout Massachusetts who are preparing students for the MCAS tests. Currently, we have over 3000 students and 50 teachers using our system as part of their regular math classes. We have had over 30 teachers use the system to create content.

Cognitive Tutor [2] and ASSISTments are built for different anticipated classroom use. Cognitive Tutor students are intended to use the tutor two class periods a week. Students are expected to proceed at their own rate letting the mastery learning algorithm advance them through

the curriculum. Some students will make steady progress while others will be stuck on early units. There is value in this in that it allows students to proceed at their own paces. One downside from the teachers' perspective could be that they might want to have their class all do the same material on the same day so they can assess their students. ASSISTments were created with this classroom use in mind. ASSISTments were created with the idea that teachers would use it once every two weeks as part of their normal classroom instruction, meant more as a formative assessment system and less as the primary means of assessing students. Cognitive Tutor advances students only after they have mastered all of the skills in a unit. We know that some teachers use some features to automatically advance students to later lessons because they might want to make sure all the students get some practice on Quadratics, for instance.

We think that no one system is "the answer" but that they have different strengths and weaknesses. If the student uses the computer less often there comes a point where the Cognitive Tutor may be behind on what a student knows, and seem to move along too slowly to teachers and students. On the other hand, ASSISTments is weak in that it does not offer mastery learning, so if students struggle, it does not automatically adjust. It is assumed that the teacher will decide if a student needs to go back and look at a topic again.

We are attempting to support the full life cycle of content authoring with the tools available in the ASSISTment system. Teachers can create problems with tutoring, map each question to the skills required to solve them, bundle problems together in sequences that students work on, view reports on students' work and use tools to maintain and refine their content over time.

Figure 1 shows how 1) students login, 2) get assignments to do, which then show up such as in the right hand side of Figure 1. Figure 2 also shows that our web-based system allows teachers access to 3) get reports, 4) manage classes, 5) get reports on students, 6a) create, edit and maintain content with the builder, 6b) find their own and others people's content (such as their students' content) 6c-e) bundling that content and assigning it to their students. We even have working reports (step 7) that automatically analyze the results of experiments that randomly assign students to conditions, which is the sort of analysis we need to determine if learning is happening.

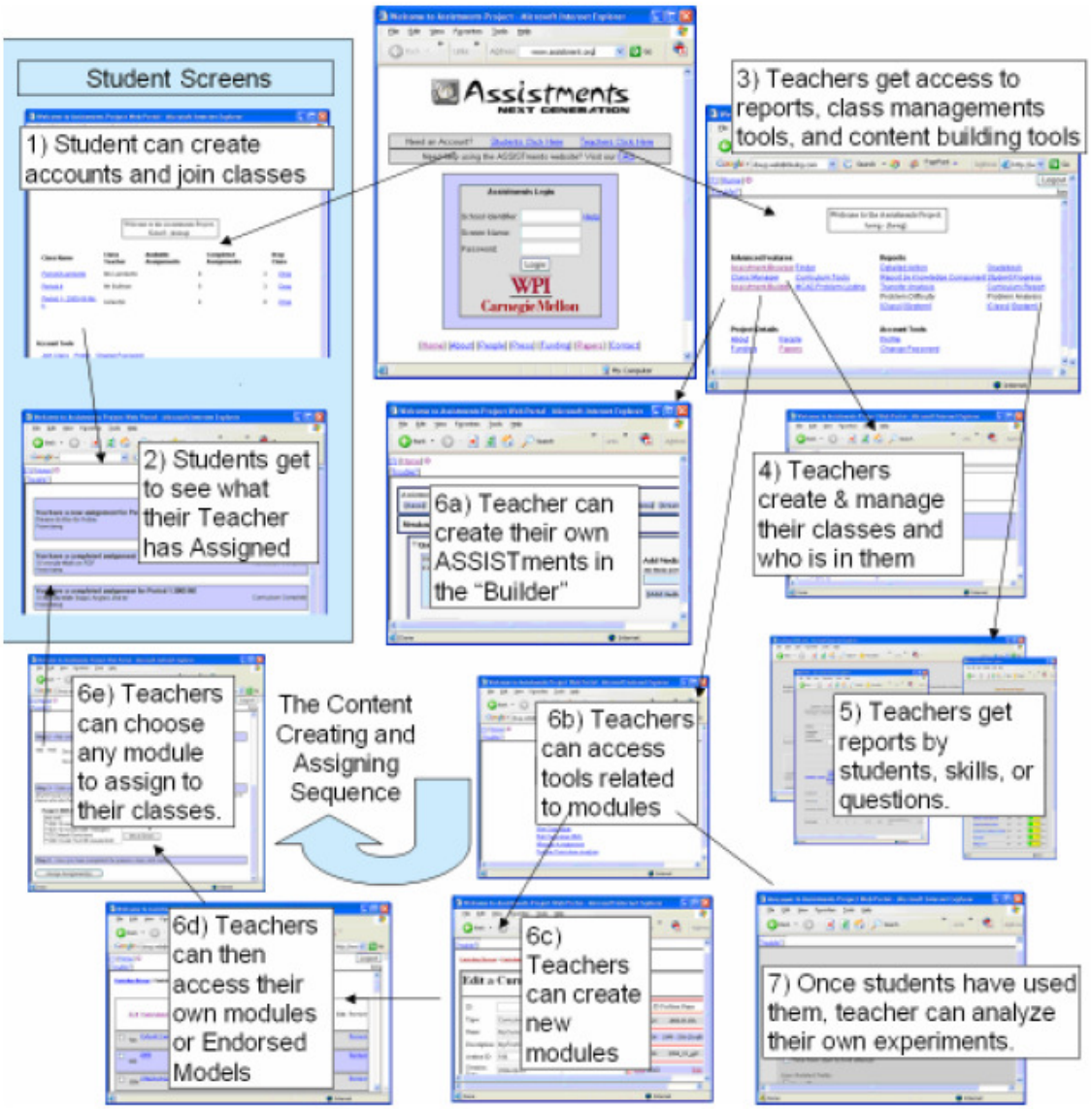

Figure 1. ASSISTments attempt to support the full life cycle of content authoring.

I.1. Structure of an ASSISTment

Koedinger et al. [7] introduced pseudo-tutors which mimic cognitive tutors but are limited to a single problem. The ASSISTment system uses a further simplified pseudo-tutor, called an ASSISTment, where only a linear progression through a problem is supported which makes content creation easier and more accessible to a general audience. Previous research has shown that our pseudo-tutor-based system can reduce the time required to build a single hour of content from 100 to 1000 hours to 10 to 30 hours [5].

An ASSISTment consists of a single main problem, or what we call the original question. For any given problem, assistance to students is available either in the form of a hint sequence or scaffolding questions. Hints are messages that provide insights and suggestions for solving a specific problem, and each hint sequence ends with a *bottom-out* hint which gives the student the answer. Scaffolding problems are designed to address specific skills needed to solve the original question. Students must answer each scaffolding question in order to proceed to the next scaffolding question. When students finish all of the scaffolding questions, they may be presented

with the original question again to finish the problem. Each scaffolding question also has a hint sequence to help the students answer the question if they need extra help. Additionally, messages called *buggy* messages are provided to students if certain anticipated incorrect answers are selected or entered. For problems without scaffolding, a student will remain in a problem until the problem is answered correctly and can ask for hints which are presented one at a time. If scaffolding is available, the student will be programmatically advanced to the scaffolding problems in the event of an incorrect answer.

Hints, scaffolds, and buggy messages together help create ASSISTments that are structurally simple but can address complex student behavior. The structure and the supporting interface used to build ASSISTments is simple enough so that users with little or no computer science and cognitive psychology background can use it easily. Figure 2 shows an ASSISTment being built on the left and what the student sees is shown on the right. Content authors can easily enter question text, hints and buggy messages by clicking on the appropriate field and typing; formatting tools are also provided for easily bolding, italicizing, etc. Images and animations can also be uploaded in any of these fields.

The builder also enables scaffolding within scaffold questions, although this feature has not been often been used in our existing content. In the past, the Builder allowed different lines of scaffolds for different wrong answers but we found that this was seldom used and seemed to complicate the interface causing the tool to be harder to learn. We removed support for different lines of scaffolding for wrong answers but plan to make it available for an expert mode in the future.

Skill mapping. We assume that students may know certain skills and rather than slowing them down by going through all of the scaffolding first, ASSISTments allow students to try to answer questions without showing every step. This differs from Cognitive Tutors [2] and Andes [18] which both ask the students to fill in many different steps in a typical problem. We prefer our scaffolding pattern as it means that students get through items that they know faster and spend more time on items they need help on. It is not unusual for a single Cognitive Tutor Algebra Word problem to take ten minutes to solve, while filling in a table of possibly dozens of sub-steps, including defining a variable, writing an equation, filling in known values, etc. We are sure, in circumstances where the student does not know these skills, that this is very useful. However, if the student knows 90% of the steps this may not be pedagogically useful.

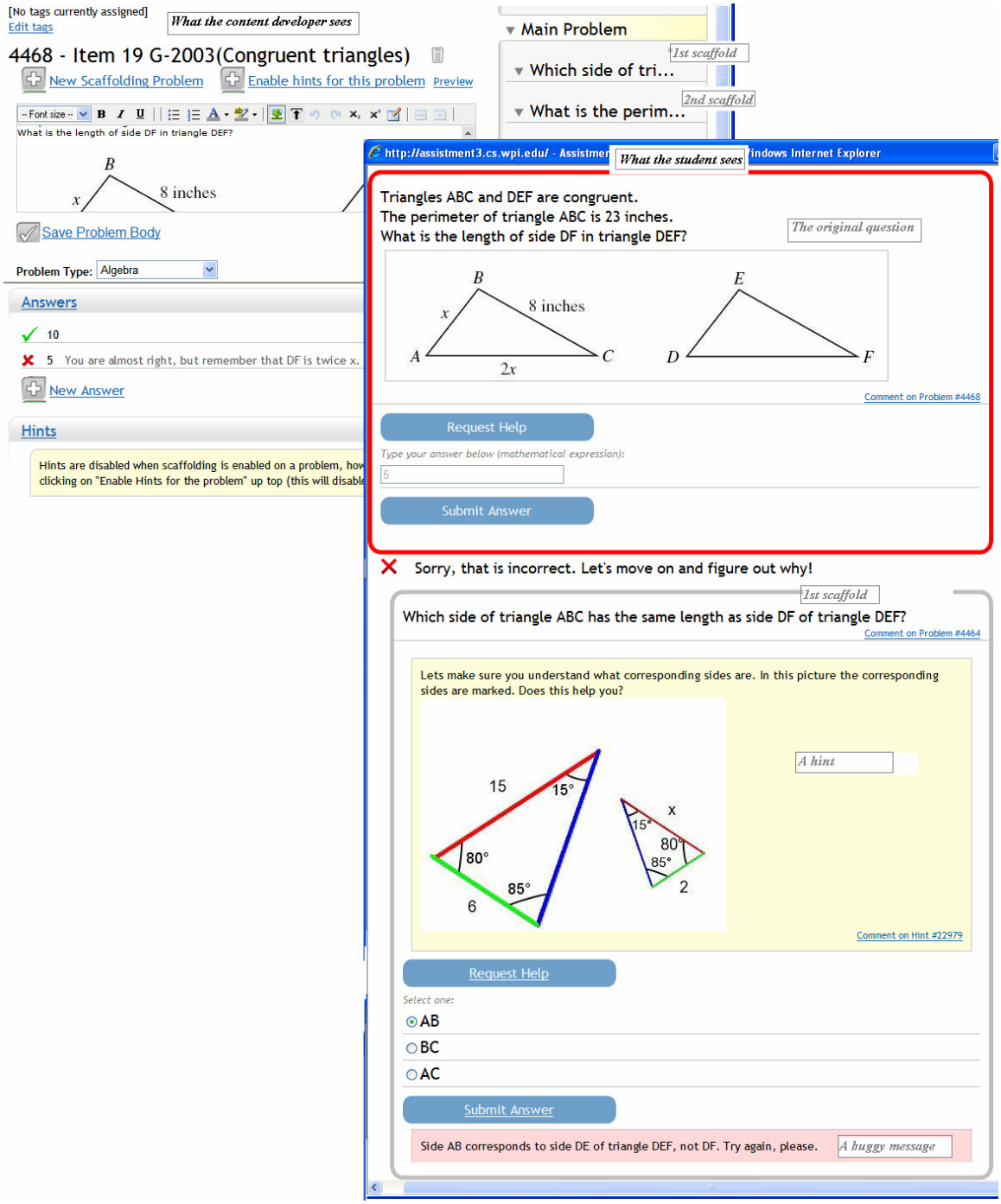

Figure 2. The builder and associated student screen

The ASSISTment Builder also supports the mapping of knowledge components, which are organized into sets known as transfer models. We use knowledge components to map certain skills to specific problems to indicate that a problem requires knowledge of that skill. Mapping between skills and problems allows our reporting system to track student knowledge over time using longitudinal data analysis techniques [4].

We currently have more than twenty transfer models available in the system with up to 300 knowledge components each. See [16] for more information about how we constructed our transfer models. Content authors can map skills to problems and scaffolding questions as they are

building content. The Builder will automatically map problems to any skills that its scaffolding questions are marked with.

I.2 Problem sequences

Problems can be arranged in problem sequences in the system. The sequence is composed of one or more sections, with each section containing problems or other sections. This recursive structure allows for a rich hierarchy of different types of sections and problems.

The section component, an abstraction for a particular ordering of problems, has been extended to implement our current section types and allows for new types to be added in the future. Currently, our section types include "Linear" (problems or sub-sections are presented in linear order), "Random" (problems or sub-sections are presented in a pseudo-random order), and "Experiment" (a single problem or sub-section is selected pseudo-randomly from a list, the others are ignored).

We are interested in using the ASSISTment system to find the best ways to tutor students and being able to easily build problem sequences helps us to run randomized controlled experiments very easily. Figure 3 shows a problem sequence that has been arranged to run an experiment that compares giving students scaffolding questions to allowing them to ask for hints. (This is similar to an experiment described in [15].) Three main sections are presented in linear order, a pre-test, experiment and post-test sections. Within the experiment section there are two conditions and students will randomly be presented with one of them.

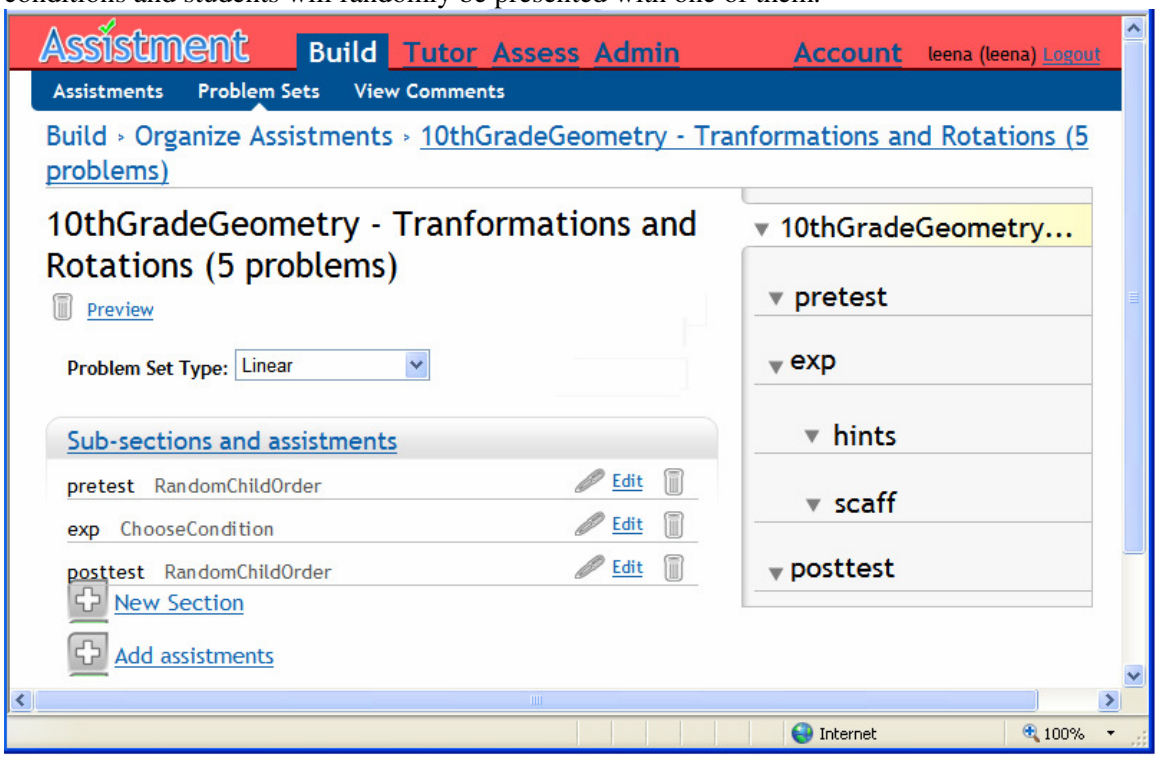

Figure 3. A problem sequence arranged to conduct an experiment.

I.3 Teacher reports

Valuable tools for teachers are the various reports that are available on their students' work. Teachers can see how their students are doing on individual problems or on complete assignments. They can also see how their students are performing on each skill. These reports allow teachers to determine where students are having difficulties and they can spend more time

on these areas. For instance, Figure 4 shows an item report which shows teachers how students are doing on individual problems. Teachers can tell at a glance which students are asking for too many bottom-out hints (cells are colored in yellow). Teachers can also see what students have answered for each question.

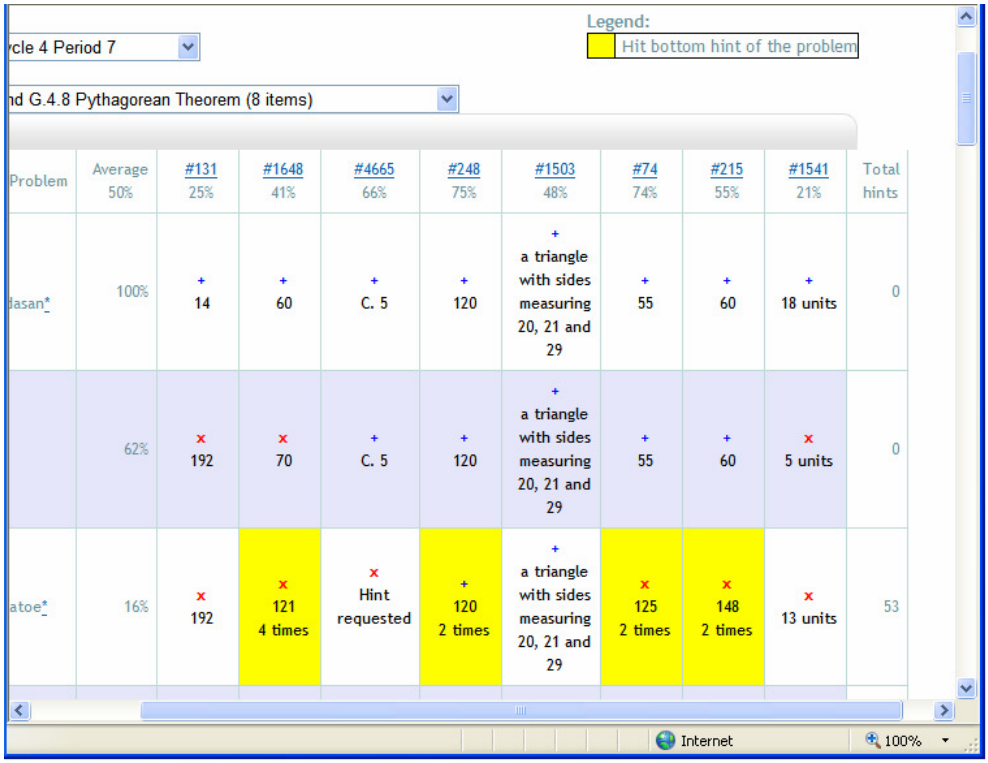

Figure 4. An item report tells teachers how students are doing on individual problems.

I.4. Experiment on cost-effective content creation in the ASSISTment system

The ASSISTment Builder's interface, shown in Figure 2, uses common web technologies such as HTML and JavaScript, allowing it to be used on most modern browsers. The Builder allows a user to create pseudo-tutors composed of an original question and scaffolding questions. In the next section, we evaluate this approach in terms of usability and decreased creation time of tutors.

Experiment methodology. We wished to create new 10th grade math tutoring content in addition to our existing 8th grade math content. In September 2006, a group of nine WPI undergraduate students, most of whom had no computer programming experience, began to create 10th grade math content as part of an undergraduate project focused on relating science and technology to society. Their goal was to create as much 10th grade content as possible for this system.

All content was first approved by the project's subject-matter expert, an experienced math teacher. We also gave the content authors a one hour tutorial on using the ASSISTment Builder where they were trained to create scaffolding questions, hints and buggy messages. Creating images and animations were also demonstrated.

We augmented the Builder to track how long it takes authors to create an ASSISTment. This does ignore the time it takes authors to plan the ASSISTment, work with their subject-matter expert, and any time spent making images and animated gifs. All of this time can be substantial, so we cannot claim to have tracked all time associated with creating content.

Once we know how many ASSISTments authors have created, we can estimate the amount of content tutoring time created by using the previously established number that students spend about 2 minutes per ASSISTment [5]. This number is averaged from data from thousands of students. This will give us a ratio that we can compare against the literature suggesting a 200:1 ratio [2].

Results. The nine undergraduate content authors worked on their project over three seven-week terms. During the first term, Term A, authors created 121 ASSISTments with no assistance from the ASSISTment team other than meeting with their subject matter expert to review the pedagogy. We know from prior studies [5] that students being tutored by the ASSISTment system spend on average two minutes per ASSISTment, so the content authors created 242 minutes, or a little over 4 hours of content. The log files were analyzed to determine that authors spent 79 minutes (standard deviation = 30 minutes) on average to create an ASSISTment. In the second seven weeks, Term B, they created 115 more additional ASSISTments at a rate of 55 minutes per ASSISTment. This increased rate of creation was statistically significant ($p < 0.01$), suggesting that students were getting faster at creating content. To look for other learning curves, we noticed that in Term A, each ASSISTment was edited on average over the space of four days, while in Term B, the content authors were only editing an ASSISTment over the space of three days on average. This rate was statistically significantly faster than in Term A. Table 1 shows these results.

Table 1

Experiment results

	Term A	Term B
Mean time to build one ASSISTment	79	55
Median time to build one ASSISTment	69	50
St. dev. on time to build	30	33
Mean # distinct days to build	4.05	3.09
Median # distinct days to build	4	3
St. dev # distinct days to build	1.28	1.86

It appears that we have created a method for creating intelligent tutoring content much more cost effectively. We did this by building a tool that reduces both the skills needed to create content as well as the time needed to do so. This produced a ratio of development time to on-line instruction time of about 40:1 and the development time does decrease slightly as authors spend more time creating content. The determination of whether the ASSISTments created by our undergraduate content authors produces significant learning is work in progress, however, our subject matter expert was satisfied that the content created was of good quality.

II. VARIABILIZATION

An important limitation of the pseudo-tutor framework used by the present ASSISTment system is the lack of ability of pseudo-tutors to generalize over similar problems [7]. A direct result of

this drawback is that separate pseudo-tutors are required to be created for each individual problem regardless of similarities in tutoring content. This process is not only tedious and time consuming, but the opportunities for errors can also increase on the part of the content creators. In our present system, about 140 commonly used ASSISTments are "morphs" – ASSISTments which have been generated by subtly modifying (e.g., changing numerical quantities) existing ASSISTments.

Pavlik et al. [14]have reported that learners, particularly beginners, need practice at closely spaced intervals while McCandliss [9]and others claim that beginners benefit from practice on closely related problems. Applying these results to a tutoring system requires a significant body of content addressing the same skill sets. However, the time and effort required to generate morphs has been an important limitation on the amount of content created in the ASSISTment system. Through the addition of the variabilization feature – use of variables to create parameterized templates of ASSISTments – to the ASSISTment builder, we seek to extend our content-building tools to facilitate the reuse of tutoring content across similar problems.

II.1 Implementation of Variabilization

The variabilization feature of the ASSISTment builder enables the creation of parameterized *template* ASSISTments. Variables are used as parameters in the *template* ASSISTment and are evaluated while creating *instances* of the *template* ASSISTment – ASSISTments where variables and their functions are assigned values.

Our current implementation of variabilization associates variables with individual ASSISTments. Since an ASSISTment is made of the main problem, scaffold problems, answers, hints, and buggy messages, this implementation allows a broad use of variables. Each variable associated with an ASSISTment has a name and one or more values. These values may be numerical or may include text related to the problem statement. For instance, the set of values of variables in a math facts type of problem may be given by {3; 7; 8} while the set of values of a variable relating to the problem statement for a problem on finding the mean may look something like this - {The population of an Aboriginal settlement over 5 years was; The time required by five swimmers to cross the strait was; The daily calorific intake of an adult is approximately}. Depending on the degree of flexibility required, mathematical functions like those to randomly generate numbers, or those doing complex arithmetic can be used in variable values. Further, we also provide the option of defining relationships between variables in two ways. The first way is to define values of variables in terms of variables that have already been defined. If variables called x and y have already been defined, then we can define a new variable z to be equal to a function involving x and y, say x*y. The other way to define a relationship is to create what are called *sets* of variables. Values of variables in a *set* are picked together while evaluating them. For example, in a Pythagorean Theorem problem, having the lengths of the three sides of a right angled triangle as variables in a set, we can associate certain values of the variables like 3-4-5or 5-12-13.

We now give an example of the process involved in generating a *template* variabilized ASSISTment and then creating *instances* of this ASSISTment. The number of possible values for the variables dictates the number of *instances* of an ASSISTment that can be generated. We can ask the builder to provide tools for generating a *template* variabilized ASSISTment by selecting the *Template ASSISTment* type as shown below in Figure 5. This causes the builder to display a widget to generate variables, as also a button called "Create Variabilized Assistments" to generate *instances* of the ASSISTment.

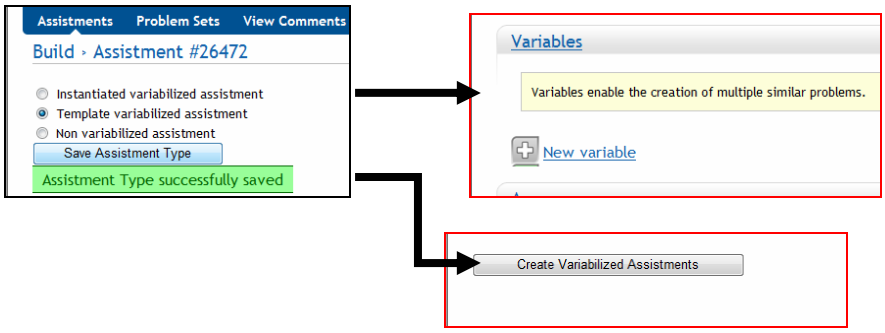

Figure 5. Choosing the ASSISTment type as a *template* variabilized ASSISTment causes the variables widget and the Create Variabilized Assistments button to be displayed.

The first step towards creating a *template* variabilized ASSISTment from an existing ASSISTment is determining the possible variables in the problem. Figure 6 shows an existing ASSISTment addressing the Pythagorean Theorem with candidates for variables highlighted. This ASSISTment is commonly encountered by students using our system and it has in all 13 hints, eight buggy messages, one main problem and four scaffold problems.

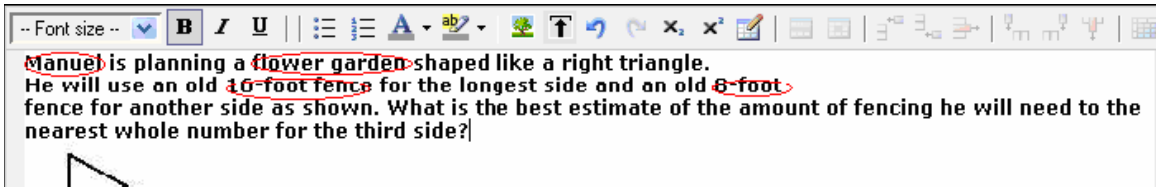

Figure 6. The non-variabilized Pythagorean theorem ASSISTment with possible candidates for variables circled.

After, identifying possible variables, these variables are created through the variables widget and used throughout the ASSISTment. As described previously, a variable has a unique name and one or more values associated with it. A special syntax in the form of ***variable-name*** is used to refer to variables throughout the builder environment. Functions of these variables can be used in any part of the ASSISTment including the problem body by using the syntax ***[function()]***. This syntax tells the builder that the function needs to be evaluated while generating *instances* of the ASSISTment. Omitting the ***[ ]*** will cause function() to merely be displayed, but not evaluated. Figure 7 shows the Pythagorean ASSISTment with variables introduced in the places identified earlier. Additional variables have been introduced to make the problem statement grammatically correct.

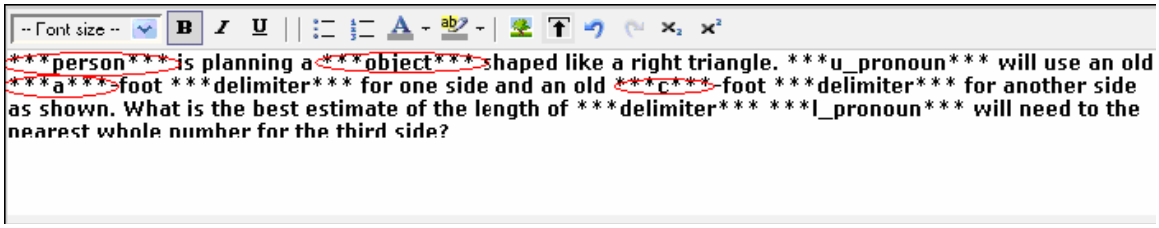

Figure 7. A sample variabilized ASSISTment on the Pythagorean theorem. As shown in red, variables have been introduced for various parts of the present problem including numerical values and parts of the problem statement.

Generation of variables in the system is simple and follows the existing format of answers and hints. Maintaining consistency with other elements of the build tools minimizes the learning time for content creators. In this ASSISTment we can make use of the *set* feature of variables to make sure that the correct values of the three sides of the triangle are picked together. For instance, in Figure 8, since the variables a, c, person, and object are in a set, the values picked for them would be 3 – 4 – Zack – house, 6 – 9 –Ming – shed and so on. The option specifying that a variable is a string tells the builder that the variable value is not a numerical quantity and should not be evaluated.

After creating variables as shown above, the same ***variable-name*** and ***[function()]*** syntax is used to reference variables and their functions in part of the ASSISTment. Figure 9 shows such a function written to calculate the answer for the Pythagorean Theorem problem and a different function used in the buggy message. This ability to define functions of variables allows content creator to calculate several intermediate values required in the ASSISTment just once and then have the system evaluate these functions for each instance of the ASSISTment created.

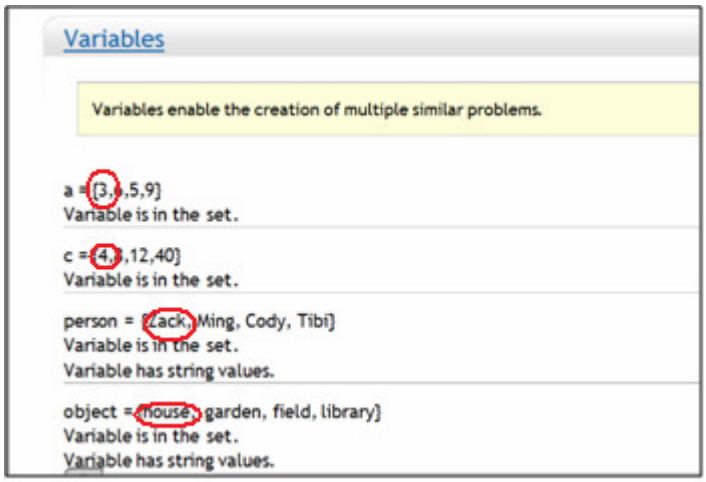

Figure 8. Generation of variables. Each variable has a name and a value. Here all the variables are said to belong to a *set.*

Once variables have been generated and introduced into problems, scaffold questions, answers, hints, and buggy messages as required, it is possible to create multiple *instances* of this ASSISTment using the *Create Variabilized Assistments* button. The number of generated ASSISTments depends on the number of values specified in the sets. Our system performs content validation to check if variables have been correctly generated and used, and alerts the content creator to any mistakes. The main advantage of variabilization lies in the fact that once a *template* variabilized ASSISTment is created, new ASSISTments including their scaffolds, answers, hints, and buggy messages can be generated instantly.

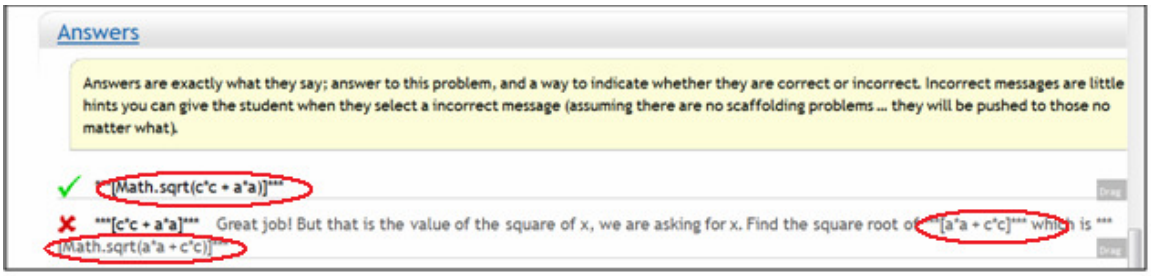

Figure 9. Variables and functions constructed from them can be used in answers and bug messages as shown.

Before creating *instances* of a *template* ASSISTments, it is also possible to preview the *template* variabilized ASSISTment. Figure 10 shows a preview of the Pythagorean Theorem ASSISTment. Some of the variables and their corresponding values have been highlighted. While the Pythagorean Theorem problem demonstrates the use of variables in small parts of the problem statement and in the numerical values, variables can be used to completely change the context of the problem.

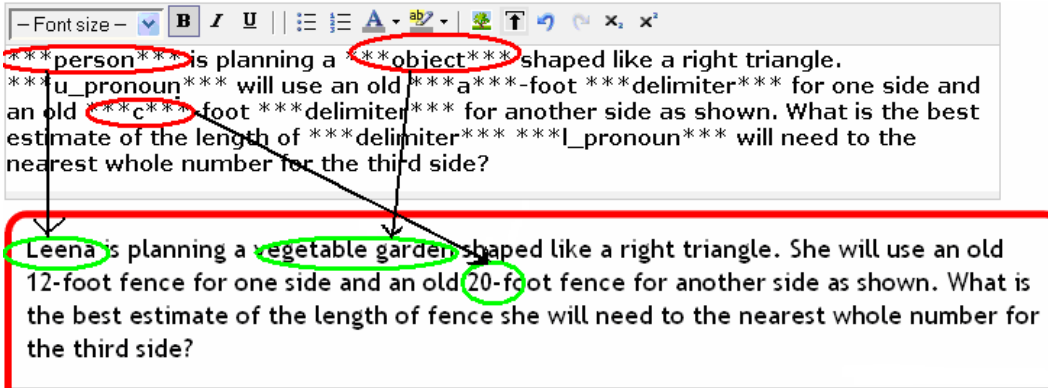

Figure 10. Preview of template variabilized Assistment. Variables and their functions are evaluated in a preview.

Our preliminary studies of variabilization comparing the time required to generate five morphs using traditional morphing techniques (e.g., copy and paste) as opposed to generating five morphs using variabilization indicate that in the former case the average time required to create one morph is 20.18 (std 9.050914) minutes while in the latter case, this time is 7.76 minutes (std 0.556776). Disregarding the ordering effect introduced due to repeated exposure to the same ASSISTment, this indicates a speedup by a factor of 2.6. Further studies are being done to assess the impact that variabilization can have in reducing content creation time. It is important to note that speedup heavily depends on the number of ASSISTments generated since creating one *template* variabilized ASSISTment requires 38.8 (std 2.783882) minutes on average as opposed to 20.18 (std 9.050914) minutes for a morphed ASSISTment. However, the variabilized ASSISTment can be used to produce several *instances* of the ASSISTment while the morph is essentially a single ASSISTment.

II.2 Variabilization for statistics tutoring

We are interested in linking the ASSISTment system to external tools that will increase its functionality. We have started to use ASSISTments at the college level in a statistics class at WPI and added functionality that supports this classroom by connecting to the R statistics program. R (http://www.r-project.org/) is a free software environment for statistical computing that includes data manipulation, calculation, and graphical display functions.

For the ASSISTment system it is very useful to make a bridge between Ruby and R. For example, we are able to support the tutoring of statistical calculations and displaying the results in a graphical format. The ASSISTment system uses RSRuby (http://raa.ruby-lang.org/project/rsruby/) which is a port of the Ruby Python module (RPy), embeds a full R interpreter and allows passing data between ASSISTment and R. To communicate with R, a Ruby object needs to be converted when it is passed to an R function. The ASSISTment system supports this conversion and communicates with R through RSRuby. Figure 11 shows the interface in the Builder. R functions can be accessed via RSRuby simply by calling a method of the same name on the RSRuby object.

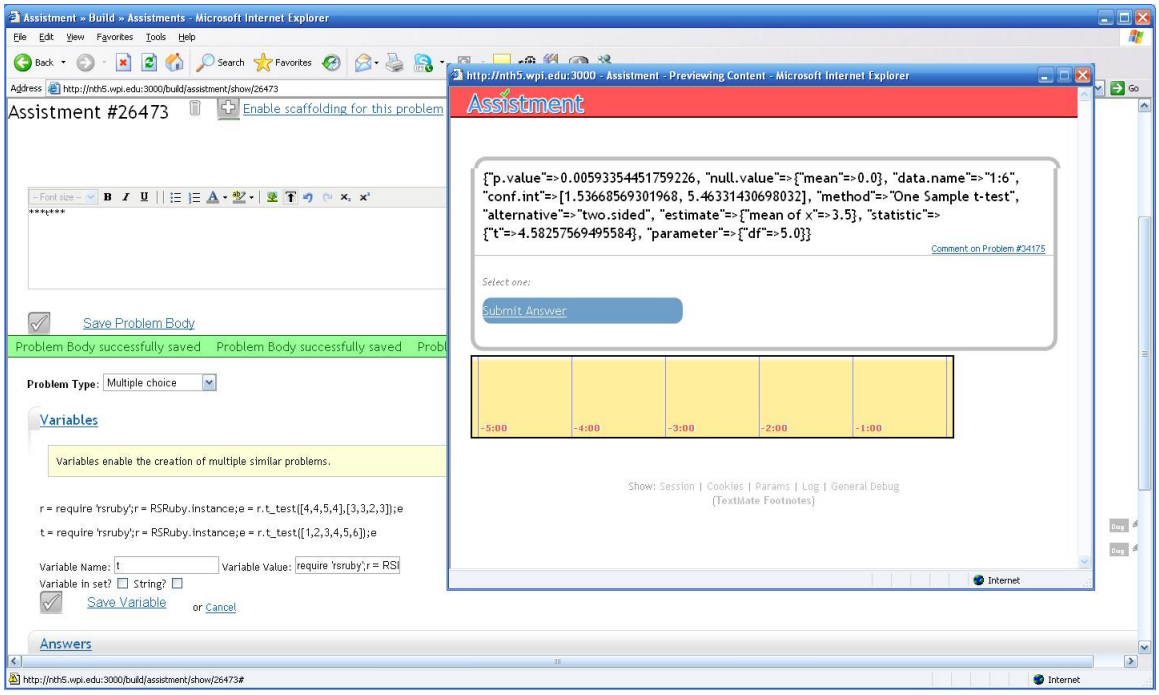

Figure 11. Builder R programming

For example, content creators can build a new problem that computes a t-test and asks for a specified return value. In the first step, authors need to create a new variable (e.g. r or t) and specify the variable value as R call. In this example, a t variable calls the t-test function of R on array [1,2,3,4,5,6]. The result is a R list that contains the p-value (0.0059), t-value (4.5825), 95% confidence interval of the hypothesis test (1.5366, 5.4633), and sample estimates (mean of x). The r variable does the same for multiple arrays. It is easy to focus on one important return value and ignore the rest of the results.

III. REFINING AND MAINTENANCE OF CONTENT

The ASSISTment project is also interested in easing the maintenance of content in the system. Because of the large number of content developers and teachers creating content and the large amount of content currently stored in the ASSISTment system, maintenance and quality assurance becomes more difficult.

III.1. Maintaining content through student comments

We have implemented a way to find and correct errors in our content by allowing users to comment on issues.

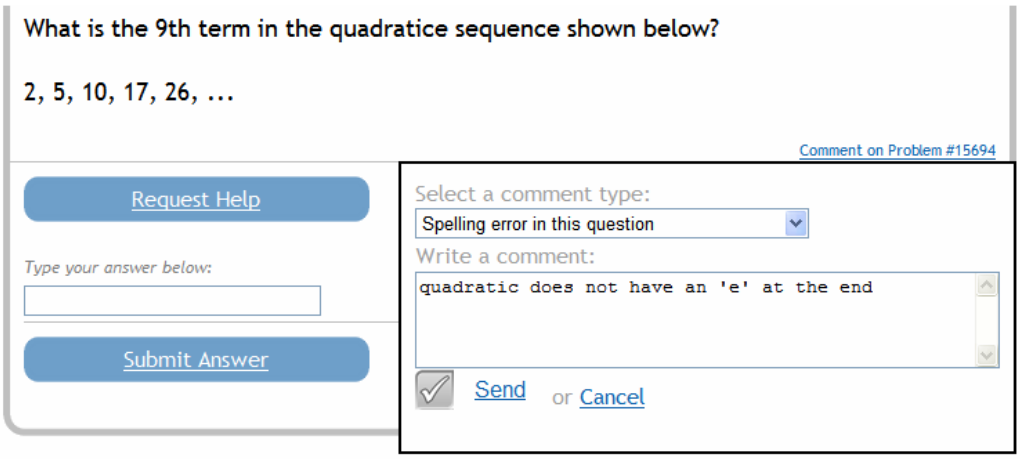

Figure 12. Students can comment on spelling mistakes, math errors or confusing wording

As seen in Figure 12, students using the system can comment on issues they find as they are solving problems. Content creators can see a list of comments (Figure 13) and address problems that have been pointed out by users.

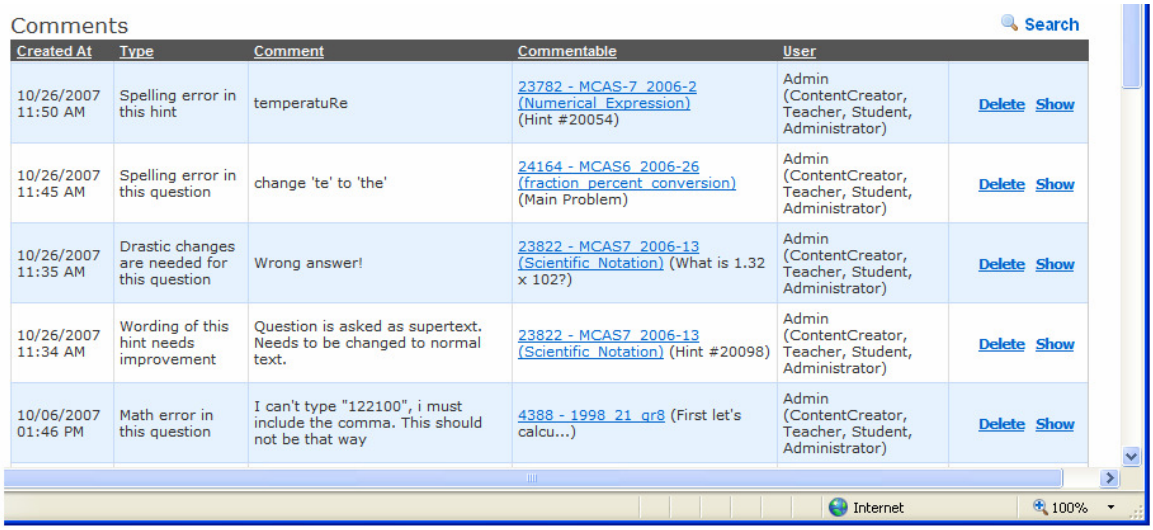

Figure 13. Authors can look through comments and resolve issues

We assigned an undergraduate student to address the issues found in comments. He reported working on these issues over 5 weeks, approximately 8 hours a week, scanning through the comments made since the system was implemented. There were a total 2453 comments, arranged into 164 pages, out of which he went through 216 comments. There were 85 ASSISTments that were modified to address issues brought up by students. Therefore, this means that about 45% of the comments that the undergraduate student reviewed were important enough that he decided to take action. We originally thought that many students would not take commenting seriously and the percentage of comments that were not actionable would be closer to 95%, so we were pleased with this relatively high number of useful comments.

Given that the undergraduate student worked for 8 hours a week addressing comments, he estimates that 80% of that time was spent editing the ASSISTments. Since he edited a total number of 102 ASSISTments (including problems brought up by professors) over the 5 week period, on average, editing an ASSISTment took a little under 20 minutes.

Many comments were disregarded because they were either repeating themselves (ranging from a couple of repeats to 20 hits), or because they had nothing to do with the purpose of the commenting system.

During his analysis, the undergraduate student categorized the comments in Table 2:

Table 2

Categorization of comments on issues with ASSISTment content

Type	No.*	Description	Brought up by:
1. Math problems	24	The information in the problem bodies did not agree with either what the hints tackle, or outdated/forgotten answers.	students and professors
2. Rewording	32	Students were complaining that some ASSISTments were wordy and confusing in the way they were written.	students
3. Broken images	22	Users complained about missing images, distorted and/or unreadable numbers in the figures.	students and professors
4. Widgets	17	Some widgets needed to be changed from *fill-in* to *algebra* or any other way to accept other pertinent correct answers or to suit better for the problem	students
5. Migration issues	10	Outdated elements from the old system: buggy messages with "null" in them, images that were above are now below, "Please select an answer" being one of the answer choices etc.	students
6. Questions mismatch	19	Original question did not match, even had different answers, in a different order or did not have the same style (multiple, fill-in)	students
7. Misspelling	15	Students paid a lot of attention to spelling mistakes.	students

*Note: a comment could have indicated more than one issue or by other comments in the ASSISTment

It was useful, when starting to edit an ASSISTment because of a comment, to find other comments related to that problem that might lead to subsequent corrections.

In addition, there was one special type of comment that pointed out visual problems from missing html code (included in the *Migration issues*). These indicated strange text behavior (i.e. words in italic, bolded, colored etc.) because of un-closed html tags or too many breaks.

In a nutshell, we believe this account strengthens the importance of the commenting system in maintaining and improving a large body of content such as we have in the ASSISTment system.

III.2. Refining remediation

There is a large literature on student misconceptions and ITS developers spend large amounts of time developing buggy libraries [19] to address common student errors which requires expert domain knowledge as well as cognitive science expertise. We were interested in finding areas where students seemed to have common misconceptions that we had inadvertently neglected to address with buggy messages.

If a large percentage of students were answering particular problems with the same incorrect answer, we could determine that a buggy message was needed to address this common misconception. In this way, we are able to refine our buggy messages over time. Figure 9 shows a screenshot of a feature we constructed to find and show the most common incorrect answers. In this shot, it is apparent that the most common incorrect answer is 5, answered by 20% of students. We can easily address this by adding a buggy message as shown below.

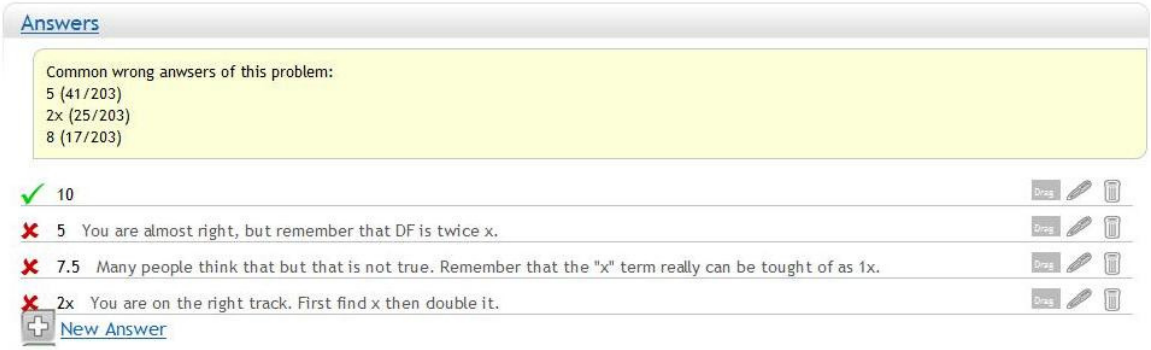

Figure 14. Common errors on particular problems are pointed

We are in the process of evaluating the usefulness of this new feature.

IV. CONCLUSION AND OPEN QUESTIONS

In this paper, we have presented a description of our authoring tool that grew out of the CTAT [7] authoring tool. When CTAT was initially designed (by the last two authors of this paper as well as Vincent Aleven) it was mainly thought of as a tool to author cognitive rules. Writing rules is time intensive. CTAT allowed authors to first demonstrate the actions that the model was supposed to be able to "model-trace". Once an author had demonstrated the rules the system could do automated testing and inform the author of what actions he demonstrated worked or did not work. The thought was that since rule writing is hard, it would be handy for a programmer to be able to be informed about how a small change he made might break other correctly working parts of the model.

It turned out that the demonstrations that CTAT would record for this automated testing seemed like good tutors sometimes, and that we might not ever have to write rules for the actions. The CTAT pseudo-tutors mimic a cognitive tutor, in that they could give buggy messages and hint messages. When funding for ASSISTments was given by the US Dept of Education, it made sense to create a new version of a simplified CTAT, which we call the ASSISTment Builder. This builder is a simplification of the CTAT pseudo-tutors in that they no longer support the writing of production rules at all, and only allow a single directed line of reasoning. Is this a good

design decision? We are not sure. There are many things ASSISTments are not good for (such as telling which solution strategy a student used) but the data presented in this paper suggests they are much easier to build than cognitive tutors. They both take less time to build and also require a lower threshold of entry (learning to be a rule-based programmer is very hard and the skill set is not common as very few professional programmers have ever written a rule-based program (i.e., in a language like JESS (http://www.jessrules.com/jess/)).

What don't we know that we would like to know? It would be nice to do an experiment that pitted the CTAT rule-based tutors against ASSISTments, give both teams an equal amounts of money, and see which produces better tutoring. By better tutoring we mean which performs better on a standard pre-test post-test type of analysis to see if students learn more from either system. We assume the rule-based cognitive tutor would probably lead to better learning, but it will cost more to get the same amount of content built. How much better does the system have to be to justify the cost? The only work we are aware of where researchers build two different systems and tried to make statements of which one is better is Kodaganallur's work [6]. They built a model-tracing tutor and a constraint-based tutor, and expressed the opinion that the constraint-based tutor was easier to build but they thought it would not be as effective at increasing learning. However, they did not collect student data to substantiate the claim of better learning from the model-tracing tutors. We probably need more studies like this to help figure out if pseudo-tutors/ASSISTments are very different from model-tracing tutors in terms of increasing student learning. The obvious problem is that few researchers have the time to build two different tutoring systems.

There is clearly a tradeoff between the complexity of what a tool can express and the amount of time it takes to learn to use a tool. Very simple web-based answering systems (like www.studyisland.com) sit at the "easy to use end" in that they only allow simple question-answer drill type activities. Imagine that is on the left. At the other extreme, to the far right, is Cognitive Tutors which are very hard to learn to create and to produce content, but offer greater flexibility in creating different types of tutors. Where do we think ASSISTments sit on this continuum? We think ASSISTments is very close to the web-based drill type systems but just to the right. We think CTAT created pseudo-tutors sit a little bit to the right of ASSISTments but still clearly on the left end of the scale.

Where do other authoring tools sit on this spectrum? Carnegie Learning researchers Blessing et al. are putting a nice GUI onto the tools to create rule based tutors [3] which probably sits just to the left of rule-based tutors. It is much harder to place other authoring tools onto this spectrum, but we guess that ASPRIRE [10], a system to build constraint based tutors, sits just to the left of Blessing's tool, based upon the assumption that constraint-based tutors are easier to create than cognitive rule-based tutors, but still require some programming.

We think there is a huge open middle ground in this spectrum that might be very productive for others to look at. The difference is what level of programming is required by the user. Maybe it is possible to come up with a programming language simple enough for most authors that gives some reasonable amount of flexibility so that a broader range of tutors could be built that would be better for student learning.

In summary, we think that some of the good aspects of the ASSISTment Builder and associated authoring tools include 1) they are completely web-based and simple enough for teachers to create content themselves, 2) they capture some of the aspects of Cognitive Tutors (i.e., bug messages, hint messages, etc) but at less cost to the author, 3) they support the full life cycle of tutor creation and maintenance with tools to show when buggy messages need to be added, and tools to get feedback from users, and of course, allowing teachers to get reports. We make no claim that these are the optimal set of features, only that they represent what we think might represent a reasonable complexity versus ease-of-use trade off.

References

[1] Aleven, V., J. Sewall, B. McLaren, and K. Koedinger (2006). Rapid authoring of intelligent tutors for real-world and experimental use. In Proceedings of ICALT 2006: 847-851. IEEE Computer Society.

[2] Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, 4 (2), 167-207.

[3] Blessing, S., Gilbert, S., Ourada, S. & Ritter, S. (2007) Lowering the Bar for Creating Model-tracing Intelligent Tutoring Systems . In Rose Luckin and Ken Koedinger (eds.) *Proceedings of the 13th International Conference on Artificial Intelligence in Education*, Los Angeles, IOS Press. pp. 443-450.

[4] Feng, M., Heffernan, N.T., & Koedinger, K.R. (2006b). Predicting state test scores better with intelligent tutoring systems: developing metrics to measure assistance required. In Ikeda, Ashley & Chan (Eds.). *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*. Springer-Verlag: Berlin. pp. 31-40. 2006.

[5] Heffernan N.T., Turner T.E., Lourenco A.L.N., Macasek M.A., Nuzzo-Jones G., Koedinger K.R., (2006) The ASSISTment Builder: Towards an Analysis of Cost Effectiveness of ITS creation. FLAIRS2006, Florida, USA.

[6] Kodaganallur, V., Weitz, R. R. and Rosenthal, D. (2005) A comparison of model-tracing and constraint-based intelligent tutoring paradigms. International Journal of Artificial Intelligence in Education, 15, 117-144.

[7] Koedinger, K. R., Aleven, V., Heffernan. T., McLaren, B. & Hockenberry, M. (2004). Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration. *Proceedings of 7th Annual Intelligent Tutoring Systems Conference*, Maceio, Brazil. page162-173.

[8] Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, *8,* 30-43.

[9] McCandliss, B., Beck, I. L., Sandak, R., & Perfetti, C. (2003). Focusing attention on decoding for children with poor reading skills: Design and preliminary tests of the word building intervention. *Scientific Studies of Reading*. 7(1) page 75 – 104.

[10] Mitrovic, A., Suraweera, P., Martin, B., Zakharov, K., Milik, N., Holland, J. (2006) Authoring constraint-based tutors in ASPIRE. Jhongli, Taiwan: 8th International Conference on Intelligent Tutoring Systems, 26-30 Jun 2006. Lecture Notes in Computer Science, 4053, Intelligent Tutoring Systems, 41-50.

[11] Mitrovic, A., Mayo, M., Suraweera, P and Martin, B. Constraint-based tutors: a success story. Proc. 14th Int. Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE-2001, Budapest, June 2001, L. Monostori, J. Vancza and M. Ali (eds), Springer-Verlag Berlin Heidelberg LNAI 2070, 2001: 931-940.

[12] Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 10, pp. 98-129.

[13] Murray, T., Blessing, S., Ainsworth, S.: Authoring Tools for Advanced Technology Learning Environment. Netherlands: Kluwer (2003).

[14] Pavlik, P. I., & Anderson, J. R. (2005). Practice and Forgetting Effects on Vocabulary Memory: An Activation-Based Model of the Spacing Effect. *Cognitive Science*. 78(4) page 559 - 586.

[15] Razzaq, L., Heffernan, N.T. (2006). Scaffolding vs. hints in the Assistment System. In Ikeda, Ashley & Chan (Eds.). *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*. Springer-Verlag: Berlin. pp. 635-644. 2006.

[16] Razzaq, L., Heffernan, N., Feng, M., Pardos Z. (2007). Developing Fine-Grained Transfer Models in the ASSISTment System. Journal of Technology, Instruction, Cognition, and Learning, Vol. 5. Number 3. Old City Publishing, Philadelphia, PA. 2007. pp. 289-304.

[17] Razzaq, Heffernan, Koedinger, Feng, Nuzzo-Jones, Junker, Macasek, Rasmussen, Turner & Walonoski (2007). Blending Assessment and Instructional Assistance. In Nadia Nedjah, Luiza deMacedo Mourelle, Mario Neto Borges and Nival Nunesde Almeida (Eds). Intelligent Educational Machines within the Intelligent Systems Engineering Book Series. 23-49 Springer Berlin / Heidelberg.

[18] VanLehn, K., Lynch, C., Schulze, K. Shapiro, J. A., Shelby, R., Taylor, L., Treacy, D.,Weinstein, A., & Wintersgill, M. (2005) The Andes physics tutoring system: Lessons Learned. In *International Journal of Artificial Intelligence and Education,* 15 (3), 1-47.

[19] VanLehn, K. (1990) Mind bugs: The origins of procedural misconceptions. Cambridge, MA: MIT Press.