WORCESTER POLYTECHNIC INSTITUTE

Facilitating WeBWorK Problem Authoring

Interactive Qualifying Project Report completed in partial fulfillment of the Bachelor of Science degree at Worcester Polytechnic Institute, Worcester, MA

Submitted to:
Professor Petruccelli (advisor)

Submitted by:
Quanquan Ma
Nathaniel Selvo
Christine Tang
Ziqing Yang

May 1, 2012

Abstract

The goal of this project was to create a graphical user interface to simplify the authoring of problems for WeBWorK, an online homework system. Interviews of potential users and research of similar user-interface applications aided in designing the interface. Tutorials for potential users were also created. Professors in the Mathematical Sciences Department at WPI tested the interface and/or the tutorials; improvements to the interface and tutorials were made based off their feedback.

Authorship Page

Quanquan Ma

Quanquan's primary role in the project was to program the logic of the application, which was used to generate the problems that can be uploaded into WeBWorK. The logic code is also responsible for importing a previously created problem back into the application.

Nathaniel Selvo

Nathaniel's primary role in the project was to program the GUI: the interface the user of the application will interact with to create a problem. Nathaniel was also responsible for implementing the logic code into the GUI, so that when the user performs an action, the appropriate part of Quanquan's code will run.

Christine Tang

Christine's primary role in the project was to create the tutorials on how to use the application, as well as create the website for the tutorials. Christine also played a key role in testing the application for errors, as she used it extensively while creating the tutorials.

Ziqing Yang

Ziqing's primary role in the project was to manage the surveying, interviewing, and testing of prospective users of our application. Ziqing was also responsible for organizing the results of these procedures.

Additionally, each person was responsible for the writing related to their role in the project. Each person was responsible for the Background Research, Methodology, Results and Discussion, and Conclusions and Recommendations part of his or her role; the other group members were responsible for editing that person's work.

Executive Summary

WeBWorK is a free open-source web-based homework system that is widely used in the United States. WeBWork has many advantages over other online homework systems; however, a major disadvantage is that WeBWorK requires problem authors to program in a specialized programming language called PG to write problems. Problem authoring can be a very tedious process, especially for those who are not comfortable with programming.

The goal of our project was to address the difficulties that problem authors have when authoring a WeBWorK problem. To achieve this goal, a java application was developed which allows a problem author to write open response, multiple choice and matching problems with the help of an easy-to-use graphical user interface. With the aid of this application, problem authors can use WeBWorK without any knowledge of the PG programming language.

Another objective of the project was performing a survey and conducting interviews with potential users before and after the application development to improve the quality of our application. A dozen responses to the survey were received, and two prospective users agreed to be interviewed. These procedures provided information about how our potential users felt about authoring WeBWorK problems in PG and what features they expected our application to have. The survey and interviews were an important part of the design of the application.

Both static and video tutorials were developed to help guide users through the process of creating a WeBWorK problem with our application. Those tutorials were then given their own website, created with Google Sites, where they could be accessed at any time by our users.

The final goal of the project was to test the application and its tutorials. Four professors in the Mathematical Sciences Department at WPI agreed to test the application while also using the tutorials that had been created. The feedback from these testing sessions was valuable, and resulted in minor changes being made to both the applications and the tutorials to improve the users' overall experience.

List of Figures

Table of Contents

1. Introduction

In education, homework assignments not only give students practice in the topics they are studying, but are also used as a measure of the student's comprehension of the material. Often, these assignments also factor into a student's final grade for the class. In the cases where the assignments are on paper, problems are assigned out of a textbook or some other source provided by the instructor. The student then completes and passes in the assignment and receives the corrected solutions days or weeks later, depending on the number of students and staff for the class. However, as with many aspects of life, computers and the internet provide methods which can automate and enhance this procedure.

When compared with paper assignments, online homework systems have several benefits for both students and instructors. For students, instant feedback eliminates the wait between submitting assignments and receiving a grade, and it allows them to address their mistakes as they make them (Gok, 2011). For instructors, automatic grading saves the time of having to collect, correct and return assignments. Automatic grading is useful for classes of any size but it is especially valuable for large classes (more than 30 students). These online homework systems also store assignments electronically, which benefits both students and instructors. Students, who often have more than one class to manage, have all their work organized for them in one place. Instructors, who have to manage the work of many students, do not have to deal with large stacks of papers.

One online homework system is WeBWorK, which was developed at the University of Rochester and is used at WPI as well as at over 240 other colleges and universities ("What is WeBWorK?", 2011). WeBWorK was first brought to WPI by Professor Farr and has been used by various classes throughout the Mathematical Sciences Department (Li and Xia, 2011, p.12). Although WeBWorK has found a place in classes like statistics and calculus (Li and Xia, p.12), there are many classes that do not use any type of online homework system. Ideally, our team would like to see WeBWorK implemented in most or all mathematics classes at WPI. Although WeBWorK was designed to be used with mathematics, courses in other departments may benefit from its features as well. Wider implementation of the WeBWorK system could improve many more classes at WPI.

Although online homework systems like WeBWorK could save time for instructors or graders in the long run, learning to use them, like any new tool, requires an initial time investment. After learning to create a class list and make a homework set, an instructor must populate the set with problems. One option is to select problems from WeBWorK's National Problem Library, but this tends to be "slow and unwieldy" (Ziemer, p.170). Additionally, the problem library only contains problems from mathematical areas. And, more importantly, they don't always target the topics or approaches an instructor wants to emphasize. The other option is for instructors to author their own problems, but this requires them to understand PG, the programming language in which WeBWorK problems are written. These difficulties in putting together homework sets are frustrating enough that they may discourage inexperienced users from even attempting to use the system. Once the author finishes the problem, he or she needs to upload the file to WeBWorK, completing a process that could be simplified to cater to new users.

Some online homework systems other than WeBWorK offer methods to lower the difficulty of authoring problems. Maple T.A. (Heck, 2006, p.15), ASSISTments ("Help/build/sequence/index", 2011) and WebAssign (WebAssign Creating Problems Guide, 2011, p.7) allow problem authors to fill out a form with fields for the question, possible choices, solution, etc. Where creating the simplest problems in these languages has authors typing into a few boxes, WeBWorK problem authors have to write a program in PG regardless of how simple the problem might be. Allowing new WeBWorK authors to experience a simpler authoring method similar to that of other systems may encourage them to continue using the system.

Our project mission was to lessen the difficulty of authoring problems in WeBWorK by implementing an interface similar to those used by other online homework systems. Accomplishing this would make it easier for professors to learn and implement WeBWorK in their courses, maximizing the benefits of the online homework system for WPI and other institutions. Our group focused on three major goals throughout the duration of our project:

- Surveying professors of the WPI community to obtain a general opinion of WeBWorK and other homework systems, as well as specific issues concerning the authoring of problems
- Designing and creating a graphical user interface for authoring WeBWorK problems which addresses the issues found during the surveying phase
- Revisiting and testing surveyed professors to measure the effectiveness of our interface, both subjectively and statistically

This process allowed our team to identify which parts of authoring WeBWorK problems are unintuitive or complex, and then create our interface to make these elements easier to perform. Having the professors we surveyed comment and test our interface showed how well the interface addresses their problems and revealed new problems which we addressed in later versions of the GUI .

2. Background

We researched the benefits and challenges of online homework systems, compared different online homework systems, and specifically focused on the advantages and disadvantages of one online homework system, WeBWorK. WeBWorK is a free, open source, online homework system currently being used at WPI and many other universities. Our project focuses on improving the WeBWorK problem authoring experience, so that problem authors can create problems more easily.

2.1. Online Homework Systems

Online homework systems similar to WeBWorK have been around for over a decade now (Bressoud, 2009). They are powerful tools that have many benefits, such as, automatic grading for instructors and instant feedback for students; however, online homework systems are not appropriate for all tasks. This section is dedicated to describing the advantages and disadvantages of online homework and examples of certain online homework systems. We will go into more details of WeBWorK in Section 2.2.

2.1.1. Benefits of Online Homework Systems

Benefits of online homework include, but are not limited to, the ability to

- give immediate feedback (Bonham, 2001)
- gather statistics for instructors (Li and Xia, 2010)
- automatically grade students’ submissions (Mendicino, Razzaq, and Heffernan, 2009)
- randomize problems among students

- allow multiple attempts for students. (Mendicino, Razzaq, and Heffernan, 2009)

Immediate feedback is one of the most appreciated aspects of web-based assessment (Gok, 2011). Some online homework systems, like WeBWorK, tell students whether they successfully solved a problem right after they submit the answer to that problem. Other systems (e.g. McGraw Hill Connect) tell students which problems they did right or wrong after they submit the whole problem set. Students don't have to wait for professors to grade their homework; therefore, they can learn right away whether they really understand a topic or not.

Professors can easily see if students are struggling with a certain topic since these systems keep track of information such as how many students have answered a certain question wrong. Some systems (e.g. WeBWorK) keep records of student's answers, so instructors can see what kinds of mistakes students are making (Li and Xia, 2010). Instructors with this information can help students more easily by taking actions such as adjusting the course schedule or holding review sessions.

Online homework systems automatically grade the problems which saves time for professors and teaching assistants (Gok, 2011). This is a significant benefit because instructors would have more time to focus on other aspects of the course (Bressoud, 2009). In turn, this can prompt students to take homework more seriously because they know it will be graded and the grade will be recorded (Mendicino, Razzaq, and Heffernan, 2009).

Instructors using online homework systems can pick from a library of problems and randomize the data for the problem or create their own problems and randomize the numbers in their authored problems. They can also randomize questions from a larger collection of problems. This randomization discourages students from copying other students' homework.

The allowance of multiple attempts, combined with the instant feedback, gives students more time to think about the problem and the concepts and formulas behind the problem (Li and Xia, 2010). Students are encouraged to continue working on problems until they get them right, or at least until they reach their attempt limit. With traditional paper-based homework, students do not generally think about problems after finishing them, since they can only get feedback when instructors return the graded papers (Li and Xia, 2010).

These are only some of the many benefits of online homework systems. Different systems have different benefits (e.g. hints, video tutorials) but the ones discussed above apply to all systems. As one can see, online homework systems have very attractive features.

2.1.2. Disadvantages of Online Homework Systems

Although online homework systems can provide benefits, they also have many disadvantages.

One of the major disadvantages is that many of the online homework systems, such as WeBWorK, do not take students' work into consideration when they require students to enter a single answer for each problem; however, problems can be broken down into multiple steps, reducing this disadvantage. As a consequence, students may be less organized because they are not required to write detailed work on paper and may choose to do more calculations in their heads. These problems can be reduced if authors choose to break the problem into multiple steps, which would also allow for partial credit to be given. Students may adopt a trial-and-error strategy that focuses only with guessing the correct answer rather than knowing why the answer is correct (Pascarella, 2004). Also, instructors may have trouble figuring out exactly where students are having difficulties without seeing their work. Finally, because students' work is not considered, it may be easier for students to cheat because they could possibly get the answers from others without having to show how they arrived at them (Mendicino, Razzaq, and Heffernan, 2009). The last problem can be addressed via randomization of homework problems, but the effort needed for randomizing will add to the workload of instructors.

Some additional disadvantages are (Jones, 2008):

- The learning experience for students who have a discomfort with working with computers may be hindered.
- The feedback may not be as detailed as paper grading.
- Some students complain that the web-based format was too dissimilar from the exams to be of help.

2.1.3. Examples of Online Homework Systems

Many online homework systems exist. We will be comparing online systems that have authoring capabilities not tied to textbooks since they relate to our project mission.

ASSISTments is a free online homework system that “ASSESS” and “ASSISTS” students by grading the work and then telling the students immediately what was wrong and helping them get the problem right (Heffernan). WPI and Carnegie Mellon University developed this tool. It is not tied to a book so students don’t have to buy an access code. ASSISTments has over 1,400 pre-made questions in its libraries, ranging from middle school math to college level calculus and differential equations; it also has libraries on science, language arts and other literature (“Libraries,” 2011, January 25). The program also allows teachers to create their own ASSISTments. The ASSISTments teacher wiki (“Help/build/sequence/index,” 2011, November 2) has online videos demonstrating how to create problems. The problem generator is easy to use but has strict input rules. The types of answers are limited to rank, fill in, check all that apply, algebra, multiple choice, and ungraded open response. The fill in problem type has a specific format (e.g. students have to type in exactly 9 versus 9.0 to get the problem right.) It has an algebra parser that uses JavaScript to evaluate expressions ("How to use Math functions," 2011, October 24).

LON-CAPA (Learning*Online* Network- a Computer-Assisted Personalized Approach) is a free open-source course management system started at Michigan State University ("Comparison of Homework Functionality"). LON-CAPA gives problem authors free access to over 300,000 electronic educational resources including about 150,000 electronic problems. The problems can be randomized with different numerical values in the same type of problem. It has about 30 different problem types. The types range from mathematical formulas to diagrammatic representation of a chemical compound. LON-CAPA also has network problem sharing and requires a Linux server (LON-CAPA).

Maple T.A. is a proprietary online testing and assessment system created by the University of Waterloo. The Maple T.A. website claims to have the easiest and most comprehensive content authoring tools available with

- *Step-by-step question designer for easy question creation and editing*
- *Over 15 question types*
- *Flexible algorithms provide a high degree of control over randomly generated questions, and can take advantage of sophisticated randomization tools found in Maple for created mathematical objects, such as matrices and prime numbers*

- *Rich editing environment supports the use of mathematical notation, images, plots, formatting, and special characters in questions* (“Why Maple T.A.?”)

Maple T.A. is powered by Maple, a computer program that allows users to analyze, explore, visualize, and solve mathematical problems, (“What is Maple”*) and is designed for mathematics, so it accepts mathematically equivalent answers and notation. It also can be connected with Blackboard to be used directly in Blackboard* (“For Blackboard Software”).

For authoring simple questions, Maple T.A. has a Graphical User Interface (GUI). For more capabilities, it uses a special script language (Alspach). Maple T.A. uses the WebEQ programmable editor interface to create customized palette questions (Heck). MathFlow Software Development Kit is replacing WebEQ (“WebEQ Developers Suite Reaches End of Life”). MathFlow provides editing, display, and accessibility of mathematical notation for websites, applications, and services.

WebAssign[1] is a proprietary online homework system. Problem authors use html for easier problems and Perl for more complex authoring (Alspach). It has its own markup language called WaTeX which is a mix of HTML and LaTeX (“Displaying Notation with WaTeX”). WebAssign can be tied to a textbook so teachers can assign problems from a textbook or access the authored collection and public collection. WebAssign also allows problem authoring (“Question Options”). And, like all the other online homework systems listed here, WebAssign can randomize numbers to create different versions of the same question (“Create Your Own Questions”). WebAssign can also be integrated with Blackboard (“Blackboard Integration”).

[1] WebAssign is a registered service mark of North Carolina State University under license to Advanced Instructional Systems, Inc.

2.2. WeBWorK

Although there are many types of online homework systems, we will be focusing on WeBWorK since it is used in many mathematics classes at WPI. WeBWorK is a free open-source web-based homework system, which was developed at the University of Rochester and is used at WPI as well as at over 240 other colleges and universities ("What is WeBWorK?", 2011). Over the past four years instructors in the Mathematical Sciences Department at Worcester Polytechnic Institute have chosen the WeBWorK system for use in calculus and statistics (Li and Xia, 2010). This section talks about the benefits and drawbacks of WeBWorK and how they compare to other online homework systems.

2.2.1. Benefits of WeBWorK

In 2009, the American Mathematical Society sent an online survey on the use of homework software to 1230 U.S. mathematics and statistics departments. Of the 467 that responded, 260 reported using such software. Among these, WeBWorK was the third most widely-used overall, and the most-widely-used non-proprietary software. Instructors like to use WeBWorK for several different reasons. Compared to other online homework systems, WeBWorK

- Is free for students and the institution
- Allows students to input mathematical expressions
- Can grade any type of answer for which it is possible to write programmed instructions to determine correctness

- Contains an extensive library of pre-written questions, the National Problem Library (NPL)
- Gives problem authors complete control over the problems they write. For example, an author can program a sophisticated algorithm to check student answers.
- Is flexible for creating new unique problems (Bressoud, 2009)
- Is "open source", meaning that there is a community of developers contributing code that has been integrated into the core of the system (Ziemer)

2.2.2. Disadvantages of WeBWorK

Although WeBWorK has many benefits, it also has issues that discourage people from adopting it. For example, some of the drawbacks of WeBWorK are that it:

- Needs to be maintained on a local server for large courses, although courses of fewer than 100 students can use the MAA WeBWorK server (AMS Homework Software Survey, p4)
- Doesn't offer video tutorials (Dreibelbi)
- Requires problem authors to know a programming language called PG, which stands for Problem Generation, to be able to write problems (Ziemer)

Our project focuses on the last challenge listed above. We plan on creating an application that would not require problem authors to know PG to author a WeBWorK problem.

2.2.3. Difficulty in WeBWorK Problem Authoring

WeBWorK requires problem authors to know how to program in PG to write problems. For this reason, WeBWorK can add to the workload of the instructor. It is possible for an instructor to spend a lot of time creating WeBWorK questions for a single class. As a consequence, instructors tend to choose a textbook with exercises or find some other source of ready-made WeBWorK questions. (Lucas, "*Using WeBWorK...", p12.*) Problem authoring can be a very tedious process, especially for those who are not comfortable with programming.

In an interview conducted by Li and Xia on September 17, 2010, Professor Farr of WPI's Mathematical Sciences Department said that WeBWorK might not be easy to start with for most instructors who were first time users. The key problem is that if instructors do not like to use existing problem sets shared online, they need to input their own homework sets or modify existing homework which may require learning the WeBWorK programming language, PG. (Li and Xia, 2010, p12)

In addition, as our group went through the official WeBWorK forum, we noticed that there have been many instructors asking for help with authoring WeBWorK problems. Also, there are many posts for WeBWorK Consultant Training Sessions which are conducted for recruiting experienced instructors as WeBWorK consultants. The consultants then would conduct WeBWorK workshops in their regions to help instructors who are interested in trying WeBWorK but would like some help in getting started.[2] Apparently, helping instructors new to WeBWorK to learn the PG language has taken the WeBWorK development team a considerable amount of effort.

[2] For example, one of the posts about WeBWorK Consultant Training Sessions can be found at http://webwork.maa.org/moodle/mod/forum/discuss.php?d=2343

2.2.4. Attempts by Others to Improve WeBWorK Problem Authoring

The American Institute of Mathematics (AIM) ("AIM07/WorkingGroups," 2008) held the Enhancing the Problem Authoring Capabilities of WeBWorK workshop specifically to address the difficulties of problem authoring in WeBWorK. The workshop was held in Palo Alto, California from August 6 to August 10 (Cervone, 2007a). It brought together 22 ("Enhancing the problem authoring capabilities of WeBWorK," 2007) like-minded individuals to figure out ways to make the PG "language easier to use, learn, document, and maintain" with "emphasis…placed on making the language easier to use for new instructors, including those with less technical expertise" (Cervone, 2007a).

The following are statements from some of the participants of the workshops; these statements are taken from "Enhancing the problem authoring capabilities of WeBWorK" cited in our References section.

Aaron Wootton wanted to learn how to be better and faster at modifying and writing problems. His major obstacle was with the PG language, which he described as "fairly complicated". He describes himself as "a low level programmer". Writing and modifying problems took him a lot of time; he also gave up many times because he did not understand the code or did not have enough time.

Bob Byerly specifically addressed the possibility of creating an interface for writing problems with simple answers (e.g. simple number answers, simple formulas and multiple-choice answers) without needing to know the PG language at all.

Davide Cervone stated that "many problem authors are hindered by the amount of perl programming that is required for writing problems. [He thinks that it] would be nice if

WeBWorK offered a problem development environment that [allows] authors to specify the details of the problem, and then [package] that up in the code necessary to make a working problem file."

Karen Clark wanted to see documentation on how to write problems in WeBWorK for novices. She stated that "it would be wonderful if a person who was [familiar with only] something as basic as equation editor on Microsoft Word…would be able to design webwork problems"

Michael Gage was interested in creating macros[3] that would make the PG language have a uniform look-and-feel; therefore making it easier to learn and use in the future.

Adolph Koski was interested in "making WeBWorK more accessible to new users". He had two main suggestions that apply to our project:

1. Centralized Training—workshops to learn how to use WeBWorK
2. Problem Creation Sites—websites that would help new users
    a. Learn the language
    b. Practice authoring problems
    c. Examine and modify sample problems

He wanted new users to practice with fragments of the language so they would not be overwhelmed or "confused with all the details at once." Koski had more suggestions but they will not be discussed since they do not relate to our project ("Enhancing the problem authoring capabilities of WeBWorK," 2007).

[3] Explanation about "macros" can be found in Section 2.3.1.

According to the workshop report, applications are being developed to help new problem authors. Davide Cervone, developer of the MathObjects portion of WeBWorK, has been developing a markdown language version of the PG language called PGML, which is easier to use for problem authors (Cervone, 2007b). A recent note (June 25, 2011) by Cervone from the official WeBWorK website shows that PGML is still in development[4].

2.3. The PG Language

The PG language, in which WeBWorK problems are written, can be broken down into three smaller entities: Perl macros, PGML commands, and MathObjects. We will talk about these in the following sections.

2.3.1. Perl Macros and PGML Commands

Perl is a programming language that was developed by Larry Wall in the late 1980s, as a way to make report processing easier. Since then, it has moved into a large number of roles, such as automating system administration and acting as glue between different computer systems (Sheppard, 2000).

A macro is a single instruction that expands automatically into a set of instructions to perform a particular task (Oxford dictionary). In the PG language, macros look for certain keywords in an author's problem code, such as "Context" or "SOLUTION". These keywords are then expanded into commands written in Perl, which PG is based on. Without these macros,

[4] The note can be found at http://webwork.maa.org/wiki/PREP_2011_Notes_June_25

problem authors would have to type multiple complex lines where each keyword would go, which is far beyond an average author's capability.

Second are PGML commands, which simplify how to code the layout of a WeBWorK problem. Using PGML commands, a user can specify the question, length of the student's answer box, and the answer which goes in that box all on the same line of code (official WeBWorK wiki). These commands provide a user interface that is more like mathematics than computer code, making it easier for those unfamiliar with computer programming to edit and create mathematical questions ("Release notes for WeBWorK 2.4.9." 2011, May 15). As mentioned in Section 2.2.4, PGML is still evolving.

2.3.2. Mathematical Expressions in WeBWorK

PG uses an object-oriented approach when dealing with formulas and equations. An author can define any equation as a MathObject, and that equation can then have useful functions performed on it. MathObjects can evaluate themselves for given values, add themselves to other MathObjects, find their own derivative and display themselves in LaTeX.

LaTeX is a document preparation system that is most often used for medium-to-large technical, scientific documents or almost any form of publishing (LaTeX Official Documentation, 2008). In WeBWorK, LaTeX is used to display mathematical symbols such as an integral sign or a square root sign. Those symbols can be written in text in a certain pattern, and LaTeX is able to recognize the pattern and generate the corresponding image. For example, in a PG source file the square root of two is written as "\sqrt{2}", but what is displayed to the student is "$\sqrt{2}$".

2.4. Organizing Files in the WeBWorK System

In this section, we will talk about how to organize a file in the WeBWorK system and how to upload files from a personal computer to a WeBWorK server.

2.4.1. The File System in WeBWorK

In the WeBWorK file system, homework sets are defined by two types of files, .pg files and def files. The .pg files are the problems of the homework set, with each file containing the code for one problem. The def files define the homework set as a whole, and this is where the instructor controls when to assign the set, when the homework set is due, and when to make the correct answers available. After specifying those times, the instructor specifies the file path for each problem (for example, "HomeworkSet1/problem1.pg") followed by the point value of the problem and the number of attempts students have for that problem.

2.4.2. Uploading a File to WeBWorK

Uploading a file to WeBWorK is similar to uploading files to other programs or websites. From the WeBWorK file manager, the user clicks the "Choose File" button and the selects a file from his or her directory to upload, then clicks the "Upload" button. However, WeBWorK only allows the uploading of one file at a time, so the user will have to compress multiple files into one file if he or she wants to expedite the uploading process. The file manager also allows the user to rename, copy, edit, delete, download, and make an archive of any file that has already been uploaded into the system.

2.5. Graphical User Interface (GUI)

User interface is the way humans interact with computers or electronic devices. There are many types of user interfaces, such as, graphical user interfaces (GUIs) and web user interfaces (WUIs). A graphical user interface (GUI) is a human-computer interface (i.e., a way for humans to interact with computers) that uses windows, icons and menus which can be manipulated by a mouse and, to a limited extent, by a keyboard.

A major advantage of GUIs is that they make computer operations more intuitive, and thus easier to learn and use. For example, it is much easier for a new user to move a file from one directory to another by dragging its icon with the mouse than by having to remember and type seemingly arcane commands to accomplish the same task.

A friendly user interface in an online homework system has many benefits. For example, Assessment, Review and Instruction System (ARIS), an online electronic homework and course management system, listed several advantages of their user interface. These main advantages include:

- Professors and students sharing the same view of the problem so there is no need for changing from instructor view to student view
- A fixed toolbar for easy navigation prevents users from getting lost; editing and creating questions are as simple as copying and pasting.

A GUI can be designed and implemented using GUI builder software, such as, NetBeans. The implementation of the GUI of our application will be discussed later in the Methodology section.

2.6. Conducting a Survey

In order to make sure our application is user-friendly, we conducted a survey before the design and implementation of our application to identify the problems of WeBWorK problem authoring, and we conducted another survey after the implementation to gauge our application's effectiveness. Before designing and conducting these surveys, we researched what steps should be taken in conducting a good survey ("Smart Survey Design").

According to the article "Smart Survey Design," the steps to conducting a survey are to:

- Clarify the purpose of the survey and identify the objectives
- Identify the people who will be involved in the design of our survey and data collection process
- Assess the resources we have and use them while designing our survey
- Decide survey methods
- Write the questionnaire (Question types, survey layout)
- Test and revise the questionnaire
- Select the sample
- Conduct the survey
- Process data (prepare the data for further data analysis)
- Analyze data
- Interpret, present and share the results
- Determine what to improve on the projects/programs/applications based on survey results.

There are four primary methods used in survey research:

- Face-to-face interview
- Telephone interview
- Mail questionnaire
- Web-based questionnaire (“Smart Survey Design”).

Advantages of these four methods are respectively:

- Face-to-face interviews: Higher response and interviewers are able to document the reasons for refusal.
- Telephone interviews: Interviewers are capable of obtaining the results quickly.
- Mail questionnaire: Surveys can be distributed without the knowledge of interviewees’ contact information
- Web-based questionnaire: Most cost effective and fastest way of distributing a survey. (“Survey Design Tutorial”).

Disadvantages of the four methods:

- Face-to-face interview: The social desirability bias may affect the accuracy of the survey results. Cost per interview is expensive.
- Telephone interview: It is hard to reach people and long or complex survey questions have to be avoided.
- Mail questionnaire: It takes a long time to get results and it is impossible to determine the reasons for refusal. (“Conducting Survey Research”)

- Web-based questionnaire: E-mails can be filtered out as SPAM (“Smart Survey Design”).

Types of questions:

- Open-Ended types: Respondents answer the questions in their own words. This type of question is good to use when asking for respondents’ attitudes, likes and dislikes, opinions and any additional comments.
- Closed-Ended types: Questions with pre-designed answers are of this type, such as multiple choice questions with either one answer or multiple answers. It’s essential that people who design the questions provide the answer choices that cover all possible answers.
- Ranked questions: This type of question is best for use when all the choices are ranked according to the level of specification.
- Rating types: It’s used when asking for respondents’ behaviors and attitudes, and beliefs (“Smart Survey Design”).

Types of measurement:

- Attitudes (What people feel)
- Knowledge (What people know)
- Behaviors (What people do or have done)
- Beliefs (What people think is true)

- Evaluation (People's perception of things are/were) ("Conducting Survey Research" 1999).

The layout of the questionnaire should be neat, clear and attractive. It is good to introduce the purpose of the survey in the introduction and later explain the group or organization conducting the survey, how the data collected will be used and may include the confidentiality information. If it is an online questionnaire, it's good to provide the estimate time to take the survey. If possible, it may be helpful to include the prizes available to participants.

2.7. Making an Effective Tutorial

Our group decided to make a video tutorial that demonstrates how to use our application. To ensure that our video would be effective, we researched how to make a good video tutorial.

First, we asked the WPI Academic Technology Center (ATC) about what type of software would be best for our tutorial. Since we would be recording a computer screen, they recommended that we use Camtasia (personal communication, February 4, 2012)), which is licensed on every WPI-owned computer. Two online sources we found, Mischook's "How to Create Video Tutorials" and DarleneVictoria's "How to Create Effective Video Tutorials," state that Camtasia is typically used to create tutorials specifically for screen captures.

After deciding what software to use, a team member attended a Camtasia training session with an ATC employee on Wednesday, February 8th, 2012. The training session consisted of how to record and edit in Camtasia. The ATC employee also gave the following tips of how to make a good video:

- Talk everything out
- Be specific

- Don’t assume that the user sees the same screen as you do. Direct them back to the view you want them to be in before telling them to do something new.
- Minimize the number of steps to do something. Don’t do things the long way or a more complicated way. Keep things simple.

We found other tips on how to make good tutorials (not necessarily video tutorials) from various online sources. Our sources were mostly bloggers or people who wanted to recommend better tutorial making because they had encountered tutorials that weren’t very good (Bart, 2007). According to Ward’s article “The Anatomy of a Great Tutorial,” “not all tutorials are created equal. Some are very good. Some are great. Still others seem to fall flat on their face and hardly seem worth the server space they’re saved on.” Because these people have watched (or even created) some bad videos, they use their expertise as authors and audience members of tutorials to help others make good tutorials.

Based on advice from different websites and an article from a scholarly journal, we compiled the following Do’s and Don’ts.

2.7.1. Do’s and Don’ts of Making a Video Tutorial

Planning

Do

- *Understand your target audience.*
- *Plan out what you want to do in the video. Know what you want the layout of your video to be. Break down steps.*
- *Create a script for your video. Have a “loose” script in which you have bullet points or key points you want to say.*

Don’t

- *Bore experts with information they already know.*
- *Intimidate novices with complicated tasks.*
- *Make a video that is very long and has too many steps*
- *Make the video seem like a movie or something with complicated stories.*

Producing

Do

- *Use proper screen capturing software.*
- *Make sure your picture quality is good.*
- *Close any programs, icons, docs you don’t want others to see.*
- *Make test recordings.*
- *Record shorter videos rather than one long one if you have a lot to address*

- *Speak clearly.*
- *Keep the video simple. Keep the video short (or break it into sections so users can just go to the section they want).*
- *Make your audience engaged in the video. The beginning of your tutorial sets the tone.*
- *Say what you are going to demonstrate and how it will benefit your audience*
- *Add some value to it so people will see how it benefits them.*
- *Record the audio separate from the video. Focus more on what you want them to learn rather than focusing on messing up your lines.*
- *Keep your dialogue simple (not everyone has the same education/vocabulary level).*
- *Use a microphone (unless you don't want sound).*
- *Have a glass of water for your throat.*
- *Speak about 25% slower than you normally would.*
- *Pause for a while if you mess up, then repeat the sentence (easier to edit).*
- *Edit later. Stopping and starting your recording wastes time and disrupts the flow of the tutorial.*
- *Imagine that you are teaching someone beside you.*
- *Use a conversational speaking style.*
- *Try to avoid words/phrases sounds like "like," "you know," "um," "ah," and edit out "pregnant-pauses."*
- *Have sound checks before each and every recording.*
- *Wear headphones with a lot of bass while recording for instant feedback of the sound.*

- *Have a recap of what you did at the end so the audience will better understand and remember what went on.*
- *Keep each segment less than 10 minutes.*

Don't

- *Move the mouse around if you aren't doing something with it.*
- *Change the screen view often. It is distracting and confusing.*
- *Read word for word from the script. Don't memorize it either. Your voice will sound monotone and people may be able to tell when you're reading from a script.*
- *Speak too close to the microphone (if you are using one) or breathe into the microphone. Stay about 3 inches away and breathe to the side.*
- *Pretend that you a presenting to a room full of people. You will sound like you are lecturing.*
- *Try to sound nerd-like or hyper intelligent.*

Editing

Do

- *EDIT!!!!*
- *Normalize the audio level.*
- *Review your recorded tutorial.*
- *Upload the video for viewing.*
- *Have people who have never used the application provide feedback on the tutorial (unless your target audience are experts in the applications).*

- *Close Caption your video.*
- *Have additional text such emphasizing key things (e.g. arrows, circles, etc.).*
- *Provide multiple formats and resolutions of your video.*

(Bart, 2007), (Bikram, 2011) (DarleneVictoria. 2009), (Mischook, S., 2007) (Oud, 2009), ("Tip 20: Six Steps to Effective Tutorials"), (Ward)

2.7.2. Making a Static Web Tutorial

The WPI library research team helped our group find academic sources for video tutorials, aka multimedia screencasts. One of the academic sources was Oud's "Guidelines for effective online instruction using multimedia screencasts," which stated that "having similar words in two formats, such as text and audio together, creates extra memory load and actually decreases learning" (Oud, 2009). Multimedia screencasts puts stress on our short-term memory because there is a lot of information (text, graphics, audio, and motion) that we need to process simultaneously; therefore, "if it isn't necessary…avoid doing a multimedia tutorial or screencast" (Oud, 2009). After reading this paper, we realized that we neglected what Oud states is "the first and most important question to ask when designing screencasts or streaming video tutorials…whether the multimedia is needed at all, or whether the instruction could be done as effectively in some other way" (Oud, 2009). After thinking about this question, we decided to make both a video and a static tutorial. Some people learn better through multimedia, while others prefer reading step-by-step instructions that a static tutorial provides.

Most of the tips for video tutorials apply to static tutorials. Static tutorials are the "How to do" articles or pages on websites that explain in words and sometimes graphics the process and procedure of some task. Since we had made a script for the video tutorial, making a static tutorial with step-by-step instructions and screenshots was not that difficult. The static tutorial also allows the tutorial user to learn (read) at his or her own pace.

3. Methodology

This chapter covers the research methods we conducted to meet our goal—to better assist WeBWorK problem authors. To do this, we created an application with a graphical user interface (GUI) for more efficient problem authoring. In order to meet our goal, we had to:

- Identify the difficulties that current WeBWorK problem authors have
- Identify the reasons why some professors at WPI do not use WeBWorK
- Identify which programming languages could be used to write our application and determine the language to be used in our application
- Identify which GUI builder to use
- Learn the PG Language, Perl, LaTeX, and the programming language chosen for the application
- Create a GUI to fit the needs of current and future problem authors
- Develop an algorithm to generate the PG code
- Implement the algorithm in the programming language and link it with the GUI
- Create tutorials for the future users of the GUI

3.1. Strategy for Identifying the Difficulties of WeBWorK Problem Authors

3.1.1. Interviewing and Surveying

In order to create a user-friendly application, we first had to survey problem authors to identify the difficulties they have with WeBWorK and figure out the reasons why some professors don't use WeBWorK. We planned to interview those who hadn't used WeBWorK and determine what we could do to encourage them to use it. We wanted to involve the potential users of our application in the design of the product.

3.1.2. Before Interviewing and Surveying

Before conducting our survey, we needed to submit a form to the WPI Institutional Review Board (IRB) informing them of our plan to interview WPI professors. Since our project research had minimal risk according to IRB standards, we filled out an exemption form instead of going through the full IRB review process. This application was sent to, received and approved by the IRB. Below is the draft sent to the IRB containing questions we intended to ask professors.

Survey

- Have you used WeBWorK in your classes before?
- Will you use WeBWorK in future classes? Why or why not?
- Do you believe that online homework helps students learn?
- Do you prefer paper homework?
- Do you know how to program in any programming languages? If yes, which ones?
- Do you feel comfortable with the WeBWorK problem creating format?
- Will you be interested in trying the GUI we plan on creating?
- What they would like to see on the GUI (layout, type of questions, etc.)

- May we quote you in our paper?

If they have used WeBWorK before, ask

- When making problem sets, did you use problems from the National Problem Library or write your own problems? If they wrote their own, follow-up with
    - What challenges did you have with the PG language?
- How did you determine how much homework you gave students?
- How frequently did you assign WeBWorK sets?
    - How many questions did you have in the WeBWorK sets?
        - How many conceptual problems?
        - How many numerical problems?

3.1.3. Interviewing and Surveying Plan

We planned to email all professors at WPI, asking for volunteers to be interviewed for our project. In the email, we explained

- Who we were
- What the project was
- The purpose of doing this project
- What we will do with survey results
- What we expected to hear from them

Since we intended that our application be used to author problems in various courses and to ensure we would have a large enough target population involved in the survey, we planned to interview professors from all WPI academic departments instead of interviewing only professors from the Mathematical Sciences Department.

To supplement our in-person interviews, we emailed all WPI faculty an invitation to participate in an online survey, using the online surveying site SurveyMonkey. Our group preferred face-to-face interviews because they would give more complete information compared to an online survey.

Before we began an interview, we abided by the instructions on our IRB application by informing the person being interviewed that

- His/her participation would be voluntary
- He/she could stop participating at any time
- Not every question required an answer
- We would not quote him/her unless given permission

3.1.4. Interviewing and Surveying Process

We sent out an email, inviting all WPI professors to participate in an online survey or a face-to-face interview; however, we received only a few responses. Only one professor asked for the face-to-face interview. We later contacted some professors in person to see if they would participate in face-to-face interviews with us and we had one more positive reply. We conducted the two interviews and recorded the results.

3.1.5. Testing Plan

To verify that our GUI actually is user-friendly, we tested professors. We emailed the professors who had stated that they were willing to test a GUI in our survey or had contacted one of the team members about testing the GUI. We also contacted the two professors whom we had interviewed.

We decided to allow the professors the option to do what they wanted to in the testing, instead of restricting them to creating our pre-selected problems. We believed that by allowing professors autonomy, they would be more willing to test our GUI.

They could test our GUI, test our static tutorials, and/or test our video tutorials. We would also be there to help guide them through creating a problem and answer any questions they had, such as, questions about the GUI or questions about uploading the file(s) the GUI creates into WeBWorK and creating a homework set.

We provided the professors with the link to the website we created for the tutorials on using this GUI. They could download the GUI from that site. We then observed them while they navigated the site, used the GUI, and uploaded the file(s) into WeBWorK, providing them assistance whenever needed.

After the professors finish authoring problems, we questioned them about

- What they liked or disliked about our application
- Whether our application convinced them to author problems in WeBWorK in the future
- What improvements we should make to our application
- What they liked or disliked about the tutorials
- What should be changed on the tutorials

3.2. Major Functionality of Our Application

Our application was designed to provide the user with a user-friendly interface which can

- Process the user's input
- Write the corresponding PG code into a .pg file
- Have different forms to make various types of problems: multiple choice, matching, and open response problems.

Aside from the main features listed above, our application has these additional features

- Provides abstractions of important mathematical symbols in LaTeX
- Supports mathematical expressions in MathObjects
- Gives the author an option to enter detailed solutions

3.3. Metaprogramming and Boilerplate Code

In order to realize these functionalities, we needed to write a metaprogram: a program that writes another program (the PG code) that defines the WeBWorK problem. Metaprogramming is a widely defined concept. In a broad sense, metaprogramming is writing code that writes code (Bartlett, 2005). Metaprogramming has many different meanings depending on the context. In our case, the generated code was relatively simple since it didn't have much logic in it. Therefore, languages that are said to be good for writing very complex metaprograms, like Python, did not have too much advantage in the development of our program.

What we were doing involved plenty of straightforward pattern finding. The pattern we were trying to find is called the boilerplate, which stands for the part of the code that is repeated many times with a limited amount of alteration. Once we successfully discovered the boilerplate code, we could then construct a corresponding template and insert the key information into the template to generate the code. Below is a flowchart of the idea of this boilerplate-finding technique.

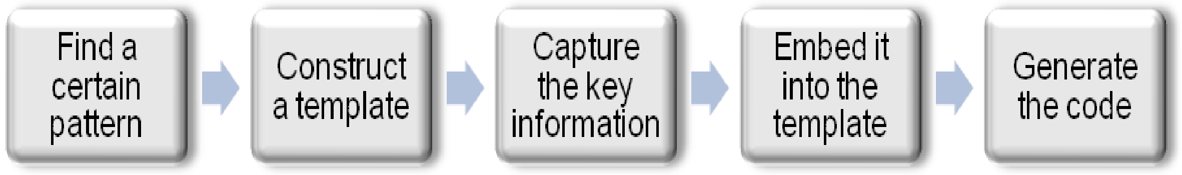


Figure 1: Boilerplate Concept Flowchart

3.4. Generating Templates

In Appendix A are two examples of the PG code for a WeBWorK problem. The key information has been highlighted. Below each code there a screenshot of what the generated problem would look like. The first, Figure 8 in Appendix A, is a fill-in-the-blank problem. The second, Figure 9 also in Appendix A, is a multiple choice problem.

The code other than the highlighted key information can be constructed as a template. The template was constructed in C-term 2012, in which we started the software development.

3.5. GUI in Our Application

After constructing the template, the next step was to get the problem information from the problem authors. A GUI was designed and implemented for this task.

In our application, we intended to design a GUI that, as part of the WeBWorK authoring process, could give a problem author an image of what the students would see. With the help of the GUI, the problem authors could easily find out where to enter the information of the problem and how to modify certain properties of the problem, such as the number of choices in a multiple-choice problem.

Below is a GUI mock-up that describes what we expected our application to look like:

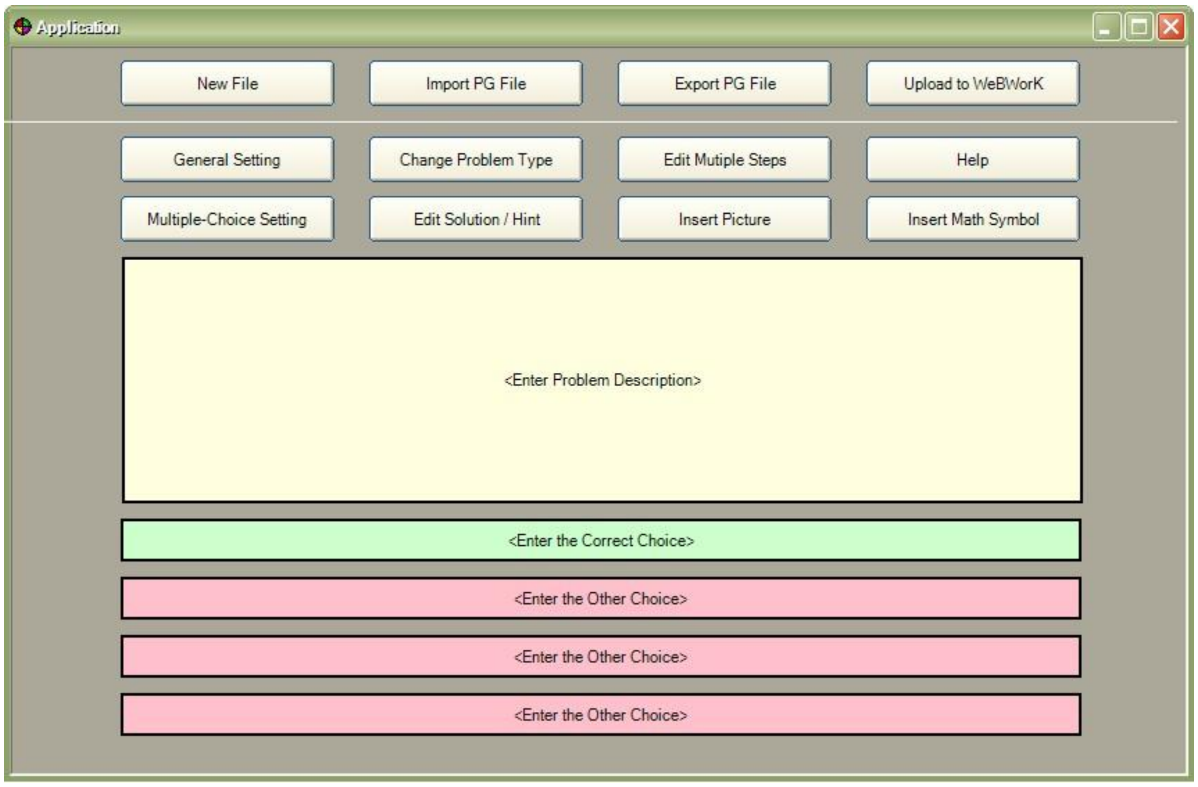


Figure 2: GUI Mock-Up

Since GUI design and GUI implementation play a significant role in our application, we needed to find a programming language that has a well-designed GUI builder and is suitable for GUI development.

3.6. Programming Languages

In order to build our interface, we needed to consider not only the language the interface would be written in, but also the programs we would use to write it. Due to the importance of GUIs discussed in the previous sections, it is important to consider that many modern languages have development environments which include GUI builders. These builders function similarly, allowing programmers to drag and drop elements (buttons, text boxes, check boxes, etc.) onto a form, which represents exactly what the user will see when the program is executed.

3.6.1. Categorizing Programming Languages

Rajaraman (1998) divides procedural programming languages into three categories (p. 47). First are algorithmic languages like C, which accomplish tasks through the use of small specialized functions. Second are object-oriented programming languages like C++ and Java, which allow programmers to define objects with attributes and methods to manipulate those attributes. Last are scripting languages like Ruby and Perl, which combine pieces of already produced code together in order to form a working program. Despite these differences, we could successfully program our interface in any of these languages, so we needed to consider them all.

3.6.2. Elegance versus Performance Trade-off

Programming languages can be ranked from low-level to high-level. The lower the level is, the more efficiently the program will run. The higher the level is, the more efficiently the programmer will program.

Machine code is generally considered to be the lowest level which directly operates the machine. Assembly language is the second lowest level which supports routines. Both of them are rarely used to program by hand because doing so can be very frustrating. They are often generated by the compiler of a higher level programming language. C and C++ are considered lower mid-level languages; they are efficient, but require the programmer to manipulate many things like memory allocation. At the next level are languages like Java, C#, and Visual Basic (VB). They provide some features to help the programmer, such as automatic garbage collection in Java, which results in relatively worse performance. Programming languages like Python, Perl, and Ruby are said to be the highest level. They provide some features like dynamic typing in Python to minimize work for the programmer . As a result, the programmer programs more efficiently but the performance is relatively the worst. We will choose a high-level language for our application, since programmer efficiency, not program performance, is our principal goal.

3.6.3. Specific Programming Languages

There are hundreds or even thousands of existing programming languages. Based on our past programming experience and our knowledge, we have listed the significant ones which we researched below.

3.6.3.1. C/C++

C++ was created as an object-oriented extension of C in the 1980's, with its final structure defined in 1998 (Al-Qahtani, Arif, Guzman, Pietrzynski & Tevoedjre, 2010, p.4). Both C and C++ have benefited greatly from their popularity, as programmers have provided many libraries and debugging tools over the last few decades (MacVittie, 1999, p. 86). Although C and C++ are some of the most powerful languages used today, they also have several faults in security. Trying to achieve a higher speed, C++ has a poor way of allocating memory while a program is running and if a programmer is not careful, the program can have an exploitable “buffer overflow”, which is quite dangerous (Al-Qahtani et al., 2010, p 24). Since we won’t need to focus on the speed, a higher-level language is clearly a better choice.

3.6.3.2. Java

Java was developed by Sun Microsystems in 1991 and since then it has gained popularity as an object-oriented language. With a structure and syntax similar to that of C++, Java has the most influence on the Internet, where programmers have the ability to create Java applets for websites. These applets can be run by anyone who has a Java Virtual Machine, a feature which is included in most web browsers, and which requires no work from the user (Al-Qahtani et al., 2010, p 12). Unfortunately, programs written in Java tend to run slightly slower than their C++ counterparts, while also using additional memory (Nisley, 2002, p.22). To reiterate, since we won’t focus too much on the speed, we can accept a slower program that’s more stable and easier to develop.

3.6.3.3. Python

Python is a high-level general-purpose programming language that can be applied to many different classes of problems. Python programs are generally expected to run slower than Java programs, but they also take much less time to develop. Python programs are typically 3-5 times shorter than equivalent Java programs (“Comparing Python to Other Languages”). This difference can be attributed to Python's built-in high-level data types and its dynamic typing. For example, when evaluating the expression a+b, it must first inspect the objects a and b to find out their type, which is not known at compile time. It then invokes the appropriate addition operation, which may be an overloaded user-defined method. Java, on the other hand, can perform an efficient integer or floating point addition, but requires variable declarations for a and b, and does not allow overloading of the + operator for instances of user-defined classes. For these reasons, Python is much better suited as a "glue" language, while Java is better characterized as a low-level implementation language ("What's Python Good for"). The latter is our focus. Ruby resembles Python very much in this aspect.

3.6.3.4. Perl

Perl was developed in the late 1980's and is useful for Unix development on the internet (MacVittie, 1999, p 88). MacVittie also states that "While Perl is easy to learn and use…it's nearly impossible even to read a complex Perl program written by anyone who hasn't toed the line of good programming practices." (p. 88). Perl tied for last in Al Qahtani's ranking of user interface development, because it lacks development tools compared to other languages (p. 88).

3.6.3.5. Visual Basic

Lastly, Visual Basic was developed by Microsoft in the early 1990's as a tool to help build programs for their own platform. It is very welcoming to new users due to its gentle learning curve, but its specialization for Windows makes it less versatile than languages like Java and C++ (MacVittie, 1999, p.90). However, this specialization also allows Microsoft to provide the necessary tools to make GUIs in Windows, as they don't have to worry about compatibility with other platforms.

3.6.4. Narrowing Down

When selecting the language for our interface, we needed to consider how the strengths of each language complement what we needed to accomplish, as well as our prior experience with each language. Out of the languages we considered, Java and C/C++ are the most versatile and the most widely used. Java is very similar to C++, which is an extension of C, but has a distinct advantage when working with multiple platforms. Java's flexibility also gives it an advantage over Perl and Python with respect to our project. With this in mind, we decided to take a closer look at Java as well as Visual Basic, because although it specializes in Windows-only programs, it was the language with which our group had the most experience.

3.6.5. Java versus Visual Basic

Java compared to Visual Basic:

- Pros:
    - Compatible in different operating systems
    - One of the best languages for web-based applications
    - Provides more security than VB
    - Easier documentation
- Cons:
    - No one in our group knows Java GUI development
    - Harder to use than VB

Visual Basic compared to Java:

- Pros:
    - One of the easiest language to learn and use
    - Better for windows application development
    - Exists mainly for GUI development
    - Two members of our team know VB already
- Cons:

- Has compatibility issues with other operating systems than windows
- Might prevent us or people in the future to integrate this application into the main WeBWorK program, which limits this application to a separate program aside from WeBWorK

We decided to use Java because its compatibility in different operating systems and its cohesion with the Internet were very beneficial in our case. Our research about other online homework systems showed that their problem authoring assisting programs were mostly written in Java.

3.7. Programming the GUI

3.7.1 NetBeans

The program we used to develop the GUI was NetBeans, a development environment for the Java programming language. Like many of the programs we considered when selecting a programming language, NetBeans has a GUI builder which streamlines the process of designing an interface. The builder allowed us to drag components (buttons, text boxes, etc.) onto a form to construct exactly what the user would see upon launching the program. As each component is placed or altered, NetBeans automatically creates or updates the code needed to display these elements.

3.7.2 Choosing the Right GUI Components

An important part of designing the interface was to select the proper components for each piece of information the user has to enter. Using the right components not only makes the GUI easier to understand, but can also guide the user in entering information. For example, things like the problem name, file path and answer(s) to each question are entered into single-line text fields because they are usually shorter in length than the question or detailed solution, which are typed into a multi-line text area.

Additionally, the components used to display information may not have the same usefulness in each type of problem. In multiple choice and matching, the answers that the user has submitted are kept in a drop-down box, which allows the user to select an element from the list, but displays only that one selected element at a time. For open response problems, the answers are stored in a list box, which like a drop-down box allows the user to select one element from the list, but displays all the elements at once. This is significant because the order of the answers in an open response problem is important, and being able to see all the submitted answers at once makes it easier to change the order of the answers if needed.

3.7.3 Programming the Components

Once all of the components had been placed and the GUI was visually complete, the final step was to add functionality to the components. Using NetBeans, programming each component consisted of right-clicking on a component, selecting an action that could be performed on it (mouseClicked, mouseDragged, keyPressed, etc.) and adding the code to execute once that action is performed by the user. For our GUI, the majority of code-executing actions occur when the user clicks on one of the many buttons, although some text fields have default text which we programmed to disappear once the user clicks into the field.

3.8. Implementing the Algorithm

We used regular expressions to embed the information. Regular expressions are a way to describe a set of strings based on common characteristics shared by each string in the set

(“Lesson: Regular Expressions”). They can be used to search, edit, or manipulate text and data. Regular expressions are the key to powerful, flexible, and efficient text processing (Friedl, 1997).

Regular Expressions were used in our application, because our application was designed to manipulate the templates that we had constructed and the templates are in a text form. There is a code package in Java called java.util.regex that was used here. The technical details of how to use Java Regular Expressions are not discussed here.

3.9. Modifying Previously Saved Work

We considered the problem of allowing users to modify existing .pg files, Such functionality would prove useful if a user needed to quit the application before the .pg file was finished, if the saved file contained errors, or if he or she just wanted to modify a file from another source. We came up with three possible solutions.

3.9.1. Importing a PG File

An ideal solution would be to allow the user to import an existing .pg file into the application to enable modification. However, this scenario would require us to summarize all syntax of the PG language and develop a program that was similar to a compiler, which would be extremely difficult even for a larger group with more time than we had.

A minor version of this solution would be to only allow the user to import a .pg file that was generated by our application. This way, we could make sure the programming style of the program was standardized so that the implementation became much simpler. Nonetheless, as a trade-off, the user would not be able to edit downloaded problems from the National Problem Library (NPL) or elsewhere, using our application.

3.9.2. Including a Database

Another solution would be to include a database in the application so we could save unfinished work as objects in the database. Upon reopening the application, the user could make a query to the database and retrieve the unfinished work as needed. This solution is not as powerful as importing a .pg file but is guaranteed to be feasible.

This solution could be extended to allow the user to save the finished job as a backup, which prevents the situation when the author has made a typo but has already exported the problem into a .pg file and has to re-enter the information again.

Since we were using the database in a relatively easy way, db4o, one of the most easy-to-use databases for Java, could be used to fulfill all the requirements.

3.9.3. Creating a User Log

The third solution would be to create a user log that records what the user had done when using our application. A user log would exist either as a giant comment in the PG file that the application would generate or as a separate text file. In this way, when a user wanted to resume the unfinished work, our application could read the user’s action records and reproduce the actions. Then the unfinished work could be reloaded into the application.

To implement user logs, we would need to develop an algorithm to write user logs as well as an algorithm to read user logs.

3.9.4. Deciding among the Methods

Table 1: Re-editing Techniques

Method	Ease of implementation	From the users' view	Extensibility	Run - time
Importing a PG file	Extremely hard, not feasible	Very intuitive, easy to use	This is the ideal situation	Slow
Minor version of importing a PG file	Hard but feasible	Very intuitive, easy to use, but would only allow the user to import a PG file generated by our program	Future teams could extend this in order to achieve the feature of importing a PG file.	Slow
Including a database	Easy, although it requires extra work to design a GUI to manipulate the database	This requires the users to do extra work to handle data in the database (such as deleting work that they don't want), which might be confusing.	Limited.	Fast
User log as a comment in the PG code	Moderate	Same as 2.	Limited.	Slow
User log as a separate file	Moderate	The user might get confused since there are two files generated.	Limited.	Slow

Method 1 was not feasible for us. Methods 3 and 5 would confuse the user which was the opposite of our goal -- an easy-to-use application. When choosing between Methods 2 and 4, we decided to use Method 2 because of its extensibility for future teams.

3.10. Tutorials

To help users of our GUI, we decided to create two types of tutorials: (1) static and (2) video. As explained in the Background Research, each has its advantages and disadvantages. To suit the preferences of different users, we created both.

For both types, we had to plan out what we wanted to go into the tutorial. First, we had to understand our target audience (problem authors). Since this GUI is a new product, we considered our audience beginners and broke down the steps to do the tasks we want them to accomplish. Then, we had to identify what the objectives of the tutorial are:

- Downloading and Opening the Application
- Selecting a Problem Type
- Creating Problems
    - Multiple Choice
    - Open Response
    - Open Response using WeBWorK's version of LaTeX
    - Matching
- Uploading into WeBWorK and Creating a Homework Set.

We created a different section for each objective (for Creating Problems we have a separate section for each type of problem) then proceeded to write an outline of what should go into each section. After creating an outline, we wrote the step-by-step instructions on how to accomplish each task. With the four problem creation objectives, a worked example was provided. How we presented the steps differed depending on the type of tutorial.

3.10.1. Static Tutorials

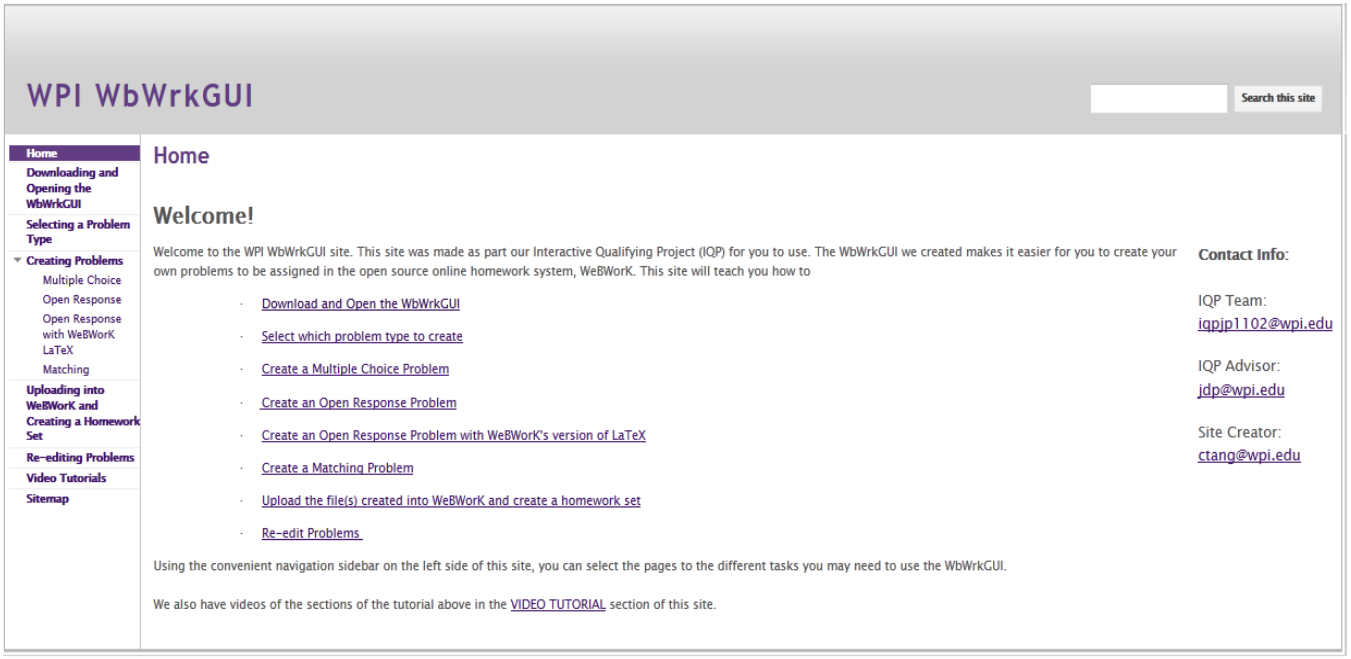
We created a Google Site for our static tutorials called WPI WbWrkGUI[5]. Google Sites are easy to use since they do not require the site creator to know HTML. Another reason we chose to create a Google Site was that one of the project team members had created a Google Site before. Editing text on the site and importing files is very intuitive for those who use Microsoft Word.

The different sections of the tutorial can be found on the sidebar on the left-hand side of the site. There is also a page for our Video Tutorial.

For each page that explains how to create each problem type, there are step-by-step instructions and screenshots of the GUI or WeBWorK site to help the user understand the steps. The instructions were written in Microsoft Word prior to creating the site. All that was required was to transfer the text in the document into the Google Site. Inserting images was less straightforward. We could not just copy and paste a screenshot in but had to save the screenshot in Microsoft Paint, add the circles and arrows to emphasize certain parts of the screen, upload the file into Google Sites, and then insert the image into the page we wanted it to be on.

For convenience, we also linked the different sections of the tutorial to the corresponding bullet points on the site's homepage. A link to the latest Java update (in case the user's version does not support the WbWrkGUI) and the file containing the WbWrkGUI were also added to the site under the Downloading and Opening the WbWrkGUI page.

[5] Link to the site: https://sites.google.com/site/wpiwebworkguitutorial/


WPI WbWrkGUI

Search this site

Home
Downloading and Opening the WbWrkGUI
Selecting a Problem Type
Creating Problems
Multiple Choice
Open Response
Open Response with WeBWorK LaTeX
Matching
Uploading into WeBWorK and Creating a Homework Set
Re-editing Problems
Video Tutorials
Sitemap

Home

Welcome!

Welcome to the WPI WbWrkGUI site. This site was made as part our Interactive Qualifying Project (IQP) for you to use. The WbWrkGUI we created makes it easier for you to create your own problems to be assigned in the open source online homework system, WeBWorK. This site will teach you how to

- Download and Open the WbWrkGUI
- Select which problem type to create
- Create a Multiple Choice Problem
- Create an Open Response Problem
- Create an Open Response Problem with WeBWorK's version of LaTeX
- Create a Matching Problem
- Upload the file(s) created into WeBWorK and create a homework set
- Re-edit Problems

Using the convenient navigation sidebar on the left side of this site, you can select the pages to the different tasks you may need to use the WbWrkGUI.

We also have videos of the sections of the tutorial above in the VIDEO TUTORIAL section of this site.

Contact Info:

IQP Team:
iqpjp1102@wpi.edu

IQP Advisor:
jdp@wpi.edu

Site Creator:
ctang@wpi.edu

Figure 3: WPI WbWrkGUI Website Homepage

3.10.2. Video Tutorials

In order to create the video tutorials, we had to develop scripts. Making video tutorials was harder and more time consuming than making static tutorials because of all the elements that would have to go into the tutorial: audio and visual. We practiced using Camtasia (screen video capturing software) in order to utilize the features of the software to best suit user needs.

For the video tutorials, we recorded the screen while we did all the tasks listed above (downloading the WbWrkGUI, creating problems, uploading the created files into WeBWorK, etc.). After the videos were recorded, we edited the videos using the editor in Camtasia and made the videos more user-friendly by adding callouts (arrows and circles) that direct the viewer to the button, link, or file they should click.

After finishing editing the video in Camtasia, we recorded the audio. Recording the audio after recording the video was easier because we just had to focus on demonstrating how to use the GUI instead of also worrying about messing up the script. We recorded the audio in Camtasia and then exported the files to Adobe Premiere to edit the audio. The WPI ATC recommended using Adobe Premiere because it is much easier to edit audio in Premiere than in Camtasia. In Adobe Premiere, we made sure that the audio corresponded to the correct part of the video. After editing and making sure the videos and audio were in sync, we went back into Camtasia to add the closed captioning, for those who prefer to read and watch a demonstration of the GUI. Camtasia has a useful tool that syncs your script to your video. All we had to do was import the script into Camtasia, hit the sync button on the closed captioning panel, play the video, and click on some of the words in the script when we heard ourselves say it. Camtasia figured out how to divide the text to readable sections for us. After adding the closed captions, we rendered the videos and uploaded them onto YouTube.

Using YouTube, we added markers to the "Selecting and Creating Problems" video. After the video plays through to the Problem Selection part of the GUI, the video stops and asks the audience to click on one of the problem types. Once the audience clicks, the video will jump to the corresponding section for that problem type. These markers make the video tutorial interactive. The audience will be actively participating instead of just watching a video.

By putting the videos on YouTube, we could more easily embed then into our website. The Video Tutorials are on the WbWrkGUI site under the page named Video Tutorials.

4. Results and Discussion

In this chapter, we present, explain and discuss the results we obtained from:

- Surveying and interviewing professors
- Writing the logic in order to write .pg files
- Creating the GUI
- Linking the GUI with the logic
- Creating the tutorials
- Having professors test the GUI and tutorials

4.1. Survey and Interview Results

Below are the results for our online survey. Since we received little feedback on this part, we did not analyze these results.

1. Have you used WeBWorK in your classes before?

4 out of 12 respondents answered "Yes". 8 out of 12 respondents answered "No".

The most common reason for those who don't use WebWorK is that they have never heard of it.

2. Do you plan to use WeBWorK in future classes?

3 out of 11 respondents answered "Yes." 8 out of 11 respondents answered "No."

All of these 3 answered "Yes" to the first question.

The most common reason for those who did not plan to use WebWorK is that they had not heard of it.

3. Do you prefer online homework or paper homework?

4 out of 12 respondents prefer online homework mostly because of the quick feedback. 5 out of 12 respondents like paper homework and online homework equally because they believe both

have advantages for student's learning. 2 out of 12 respondents prefer paper homework because they feel it is more convenient for the problems they assign, which may require students to draw diagrams or perform long computations. One respondent selected "No basis for comparison".

4. Please check the computer languages you have used before.

Out of the 12 respondents, 6 professors stated that they had used LaTeX, 5 had used C and C++; 0 had used the PG language, which implies that those who had used WeBWorK had used pre-written problems.

For those who have used WeBWorK before: (question 5-7)

5. Please select the following activities that you have done before.

"Created your own problems in WeBWorK" - Two responses

"Used problems from WeBWorK's National Problem Library" - Three responses

"Modified problems" - Three responses

"Used already created problem sets" - Two responses

It is worth noting that although no one responded that they had used PG before, two respondents said they had created their own problems in WeBWorK, which requires use of the PG language. For Question 7, none of the three respondents said that they had trouble with PG, though two respondents added additional information of "Don't know" and "Unknown". It is possible that there may have been confusion as to what the PG language is, and the respondents may have thought it was something dissimilar to what they had already experienced when using WeBWorK. Alternatively, the respondents may not have been as familiar with PG as they were with WeBWorK or the more popular programming languages listed for Question 4, and felt that it was not appropriate to select it.

6. Do you like the WeBWorK problem creating format?

Of the three respondents who answered this question:

1 out of 3 respondents answered “Like.” 2 out of 3 respondents answered “Indifferent.” No respondents answered “Dislike.”

7. Did you have trouble with the PG language?

Those who had used WeBWorK before responded that they had had no trouble with PG.

8. Would you be interested in using a Graphical User Interface (GUI) that would not require knowledge of the PG language?

6 out of 9 respondents answered “Maybe.” 1 out of 9 respondents answered “Yes.”

9. Would you be willing to test such a GUI?

4 out of 7 respondents answered “Maybe.” 1 out of 7 respondents answered “Yes.”

10. What type of layout, questions, features, etc. would you like to see on this GUI

Not enough feedback.

We interviewed two professors: Professor Tashjian and Professor Petruccelli. Professor Tashjian had never used WeBWorK for a class before, while Professor Petruccelli has. Both plan to use WeBWorK in the future. They both stated that the instant feedback for students and the automatic grading feature are key reasons why they want to use WeBWorK.

Professor Petruccelli has authored problems in WeBWorK before; he learned how to author problems by taking problems from the National Problem Library, studying the code and modifying the code to create the specific type of problem he wanted. He stated that he is “not a huge fan” of the PG language because it is “not natural” to him and the creating problem sets part of WeBWorK is “unnecessarily complicated” (personal communication, February 6, 2012).

Both expressed interest in testing the GUI. Professor Tashjian wanted the layout of the GUI to be “very simple” because she “[doesn’t] like cluttered” and likes “a lot of blank space on the screen” (personal communication, January 31, 2012).

Professor Petruccelli wanted to be able to create multiple choice, matching, True/False, and fill-in the blank problems; he also would like to be able to randomize problems so a student would have a different problem than the person whom he or she is sitting next to.

The main reason we performed the survey and interviews was to gather data on what potential users would want to see implemented into our application. Unfortunately, the survey results provided little information for us to work with. Most respondents had not used WeBWorK before, and those that had were unable to give much insight on how our application should look and perform. However, the interview results proved more useful.

By taking what Professor Tashjian had mentioned about the interface being simple and uncluttered and comparing it to our mock-up GUI, we realized that we needed to make some changes to our design. The main problem with our mock-up GUI was that it was trying to implement too many features on a single screen.   Additionally, each of the buttons on the mock-up would have initiated a complex task (changing the problem type, editing the setting for the problems, etc.) in a different window, bouncing the user back and forth between multiple windows to create a single problem. To correct this, we decided to have a dedicated window for each problem type, wiht all information for the problem to be entered into this dedicated window.

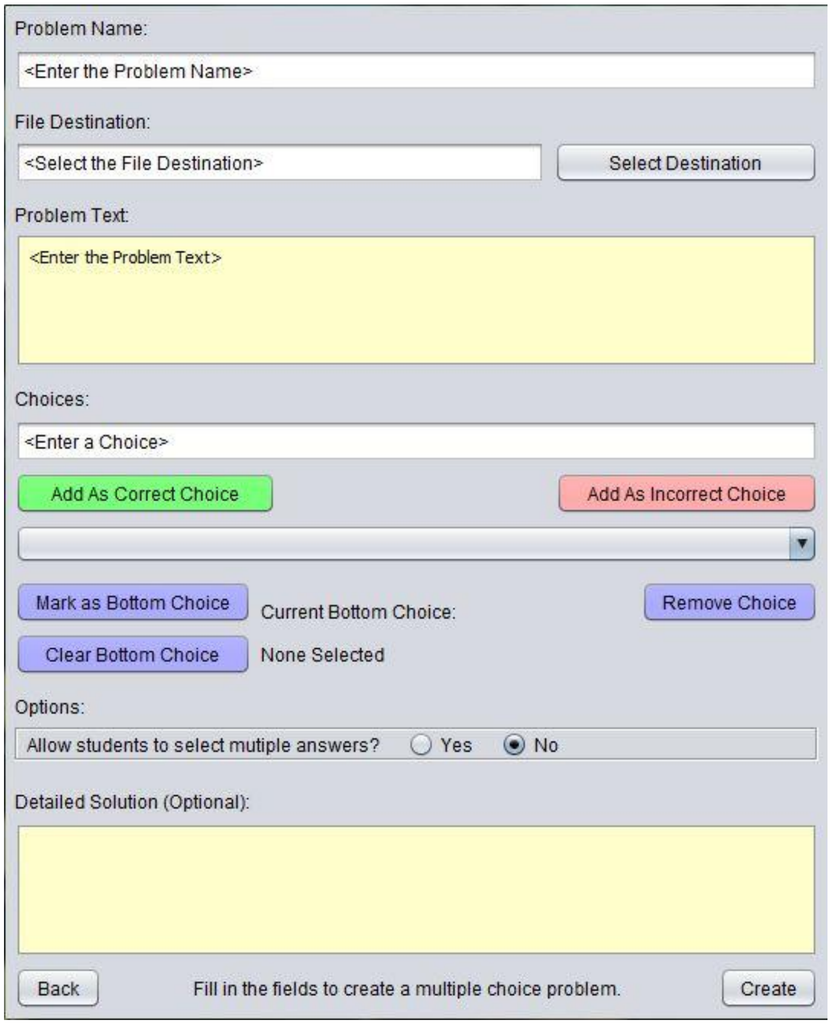
After updating the design of the GUI's windows and navigation, we needed to decide which types of problems our application would create. In his interview, Professor Petruccelli had mentioned that he wanted the application to create multiple choice, matching, true or false, and fill-in the blank problems. We agreed that all of these problem types would be possible to implement into our application, and that these types would give our users a reasonable amount of flexibility in the problems they could create. After realizing that true or false questions could be considered a subset of multiple choice problems, we concluded that our application would create multiple choice, open response and matching problems.

4.2. Graphical User Interface

4.2.1. Graphical User Interface Development

Implementing the logic code inside a user interface resulted in a program that allows users to create a variety of WeBWorK problems. On the next pages are screenshots of the GUI we created, which we named the WbWrKGUI.

Multiple Choice:



Problem Name:

<Enter the Problem Name>

File Destination:

<Select the File Destination>

Problem Text:

<Enter the Problem Text>

Choices:

<Enter a Choice>

Current Bottom Choice:

None Selected

Options:

Allow students to select mutiple answers? Yes [✓ No]

Detailed Solution (Optional):

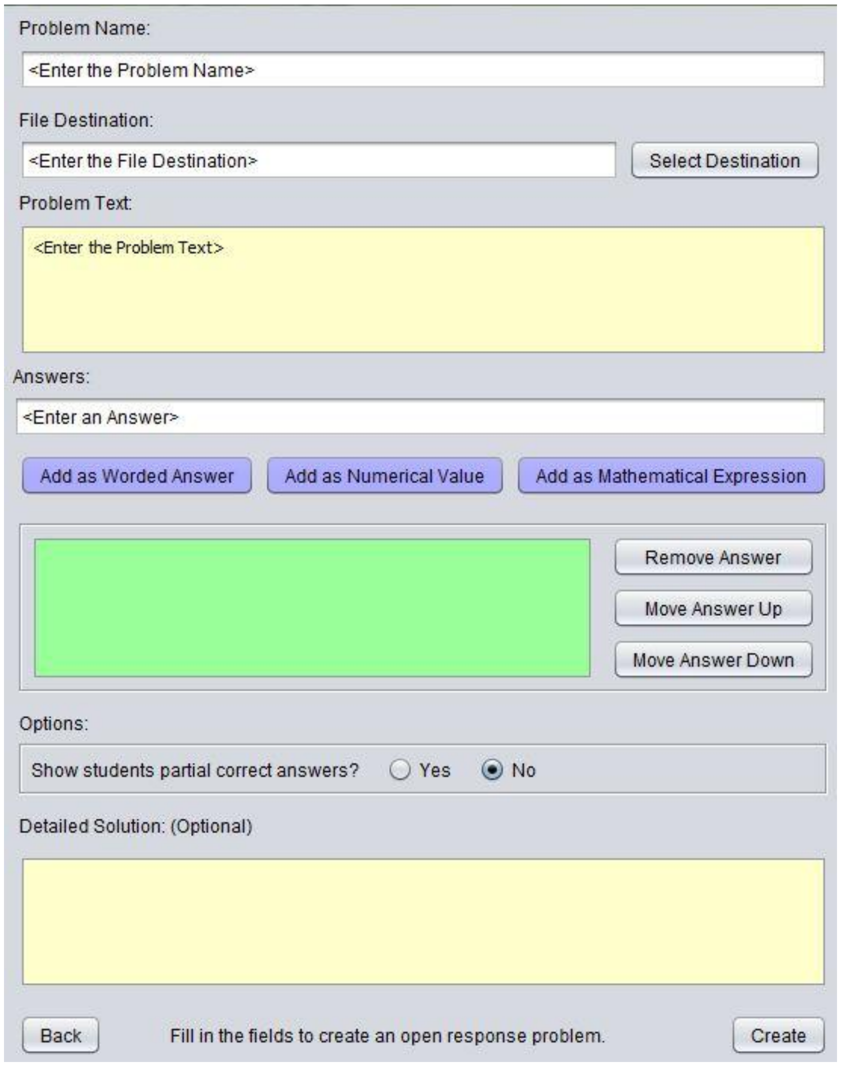
Fill in the fields to create a multiple choice problem.

Figure 4: WbWrkGUI Multiple Choice Form

For multiple-choice problems, our GUI allows users to:

- Create a question with multiple correct answers
- Add any number of incorrect answers to a question
- Set any choice to always appear at the bottom of the answer list (such as “None of the above”)
- Allow students to select a single answer (radio buttons) or multiple answers (check boxes) at a time.

Open Response:



Show students partial correct answers? Yes [✓ No]

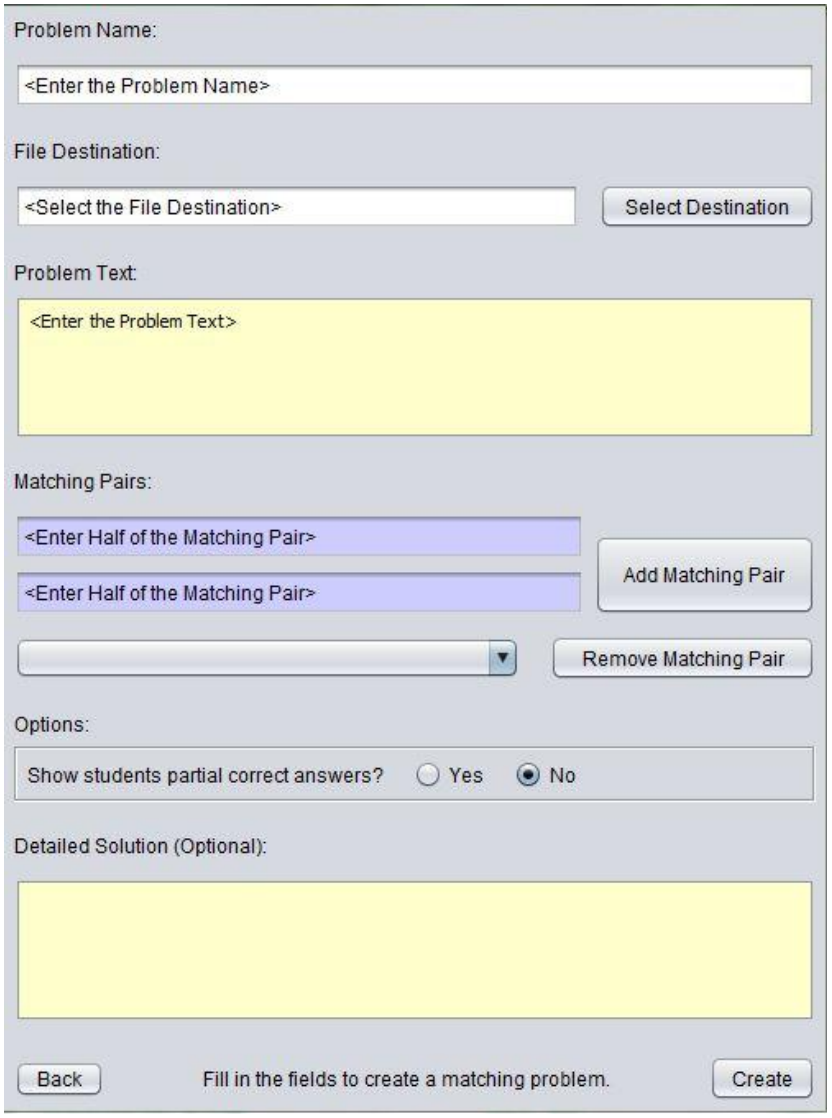
Fill in the fields to create an open response problem.

Figure 5: WbWrkGUI Open Response Form

For open-response problems, our GUI allows users to:

- Create a problem using any number of the following as answers:
  - Worded answers
  - Numerical values
  - Mathematical Expressions
- Allow students to see partial correct answers for each problem.

Matching:


Problem Name:

<Enter the Problem Name>

File Destination:

<Select the File Destination>

Problem Text:

<Enter the Problem Text>

Matching Pairs:

<Enter Half of the Matching Pair>

<Enter Half of the Matching Pair>

Options:

Show students partial correct answers? Yes [✓ No]

Detailed Solution (Optional):

Fill in the fields to create a matching problem.

Figure 6: WbWrkGUI Matching Form

For matching problems, our GUI allows users to:

- Create a problem with any number of matching answer pairs.
- Allow students to see partial correct answers for each problem.

Additionally, for all problem types, users can:

- Provide a detailed solution for each problem
- Import information from a previously created problem

4.2.2. Programming Difficulties

The most difficult thing we had to overcome for the logic part of the programming was the re-editing feature. The regular expression package, which would have been used to identify and interpret patterns in the .pg files, didn't work as expected. We switched to an alternative approach, which uses functions that manipulate strings of characters. Also, since the templates were imported line by line, it was difficult to parse multi-lined texts, so we introduced a variable called *import mode*. In this way, when a special character sequence comes up as the beginning sign of a multi-lined text, we can change the *import mode* variable to a certain value and when importing the next line, the program will know what to do with it.

Another significant difficulty encountered while programming the GUI was getting the application to generate problem files properly. The application worked properly when tested on the computer it was created on, but would only generate blank files when used on other computers. The issue was that the application was unable to find the template files needed to create the .pg files.

For each problem type (multiple choice, matching, open response), a text file containing a template for how that type's .pg file should look is included along with the code. When the application attempted to create a .pg file it would use this line of code

```
FileInputStream fstream = new FileInputStream(template);
```

to find the template using a specified file path. However, this file path is specific to one computer, so once the application was downloaded to other computers, it would search for the template, fail, and generate a blank file.

To correct this, the above line of code was changed to

```
InputStream fstream = logic.HwkProblem.class.getResourceAsStream(template);
```

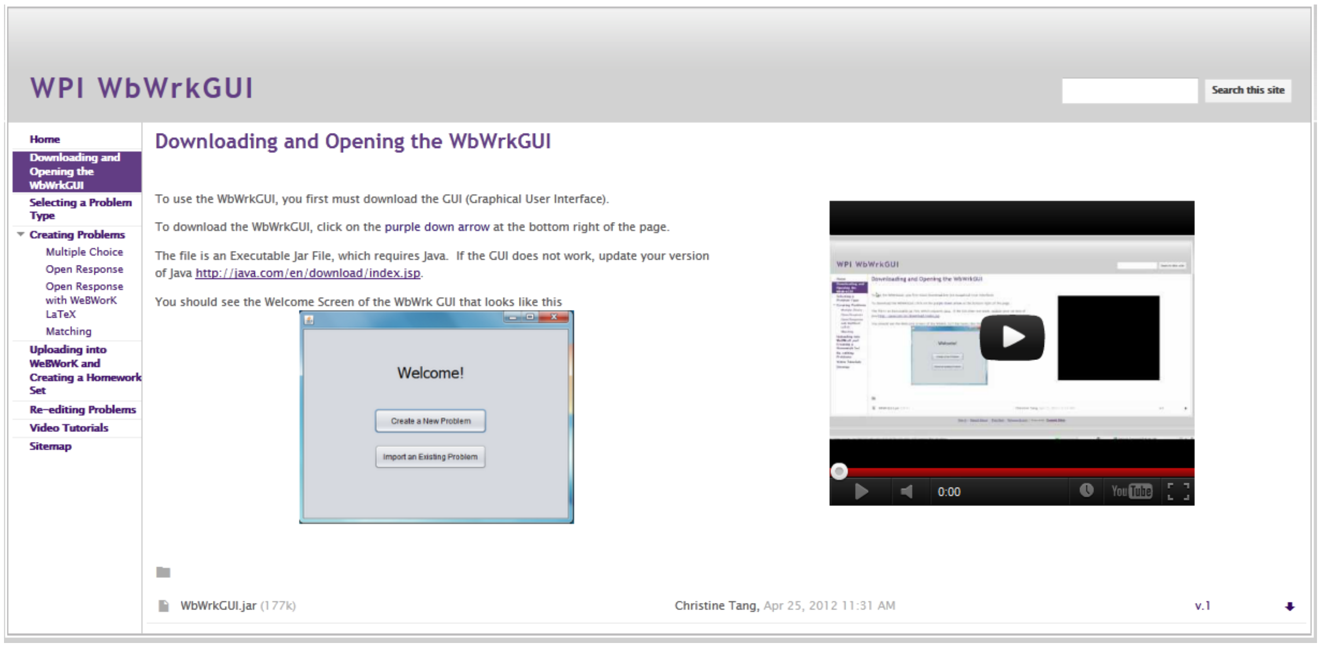
which searches for the template based on where the application is run from, allowing the application to find the template on any computer and work as intended.

4.3. Tutorials

4.3.1. Static and Video Tutorials

We created two types of tutorials: static and video. Both types can be found on the site we created[6]. The html files to the website and the mp4 video files were submitted along with this report. Below is a screenshot of a static tutorial we created, with its corresponding video tutorial embedded on the right of the page.

[6] The static tutorials are the different pages on the website (https://sites.google.com/site/wpiwebwork guitutorial/home). The link to the page containing the video tutorials is: https://sites.google.com/site/wpiwebworkguitutorial/video-tutorial



Downloading and Opening the WbWrkGUI

To use the WbWrkGUI, you first must download the GUI (Graphical User Interface).

To download the WbWrkGUI, click on the purple down arrow at the bottom right of the page.

The file is an Executable Jar File, which requires Java. If the GUI does not work, update your version of Java http://java.com/en/download/index.jsp.

You should see the Welcome Screen of the WbWrk GUI that looks like this

WbWrkGUI.jar (177k) Christine Tang, Apr 25, 2012 11:31 AM v.1

Figure 7: WPI WbWrkGUI Website Downloading and Opening the WbWrkGUI page

We created a website that contains multiple pages with different content that address different objectives:

- Downloading and Opening the WbWrkGUI
- Selecting a Problem Type
- Creating Problems and individual pages for
  - Multiple Choice
  - Open Response
  - Open Response with WeBWorK LaTeX
  - Matching
- Uploading into WeBWorK and Creating a Homework Set

The "Downloading and Opening the WbWrkGUI" page explains how to download the GUI which is at the bottom of that page. It also has a link for users to update to the latest version of Java; Java is necessary for using our GUI. The page also has the video tutorial showing how to download the application (as seen in Figure 7).

“Selecting a Problem Type” explains where to click to select a problem type.

In “Creating Problems,” we demonstrate how to create each problem type (Multiple Choice, Open Response, and Matching) through an example, providing step-by-step instructions with screenshots. We also have an example using LaTeX syntax that WeBWorK will recognize.

The second to last page (the last page being the site map) on the site contains all of our video tutorials. We have four video tutorials: “Downloading and Opening and GUI,” “Selecting and Creating Problems,” “Uploading Problems and Creating a Homework Set,” and “Re-editing.” The videos contain roughly the same material as the static tutorials.

We edited over half an hour of footage into those four videos that are of lengths 35 seconds, 7 minutes and 12 seconds, 3 minutes and 6 seconds, and 1 minute and 31 seconds respectively. The three longer videos have closed captioning.

The site and videos are currently only available to those who have the link to our website. In the future, these materials will be open for people to use.

4.3.2. Tutorial Difficulties

The only difficulties that arose from the creating the tutorials were from the video tutorials. Specifically, recording the audio for the videos took three days longer than the expected two days. The first microphone used was not satisfactory so we had to borrow another microphone. Originally, we edited audio in Camtasia but it was not very efficient and also created cuts in the video. We decided to switch to Adobe Premiere which made editing the audio much easier and did not affect the video. Embedding the videos on the website we created was also a challenge because Google Sites only allow certain types of videos to be embedded (YouTube, Google, and Google Docs Videos).

4.4. Testing Results

From our testing, we received positive feedback on the usefulness of our application. We tested four different professors in the Mathematical Sciences Department at WPI on four different days. The first two professors did not test the GUI with the re-editing feature.

The first professor we tested stated that he was very satisfied and was enthusiastic about being able to create his own problems. He had used WeBWorK in his classes before but did not author any problems.

We observed that he had trouble with uploading the file and adding it to an existing homework set; therefore showing that tutorials or user guides are needed to help future users with the steps to take to problem author.

The professor recommended:

- changing the size of a picture in the static tutorial for “Downloading and Opening the GUI”
- changing the text of a button on the Open Response Problem creation from “Add as Numerical Expression” to “Add as Mathematical Expression” because it was confusing to him since we had another button with the text “Add as Numerical Value.”

The second professor we tested was Professor Tashjian, whom we interviewed earlier in this project. She was not familiar with using WeBWorK and had never created her own

WeBWorK problems before. She first followed our instructions on using the application, but since the process was confusing to her, one of our teammates showed her the steps by clicking the buttons for her. She was very impressed with our application and she thought it was very helpful and useful for her. She even stated that our application was very fantastic and “there was nothing like it in the previous WeBWorK.”

She suggested that it would be better if we could make it applicable for students to use the integral sign in the answer box. As for now, only problem authors could use it when they created their problems.

The third professor we tested is relatively familiar with WeBWorK. He had never created his own WeBWorK problems before. He was the first professor who also tested our video tutorial and he thought the video tutorial was fine. Then he went through the static tutorials, followed the guidelines and successfully completed most of the steps. He was also satisfied and happy with our application. He thought it was useful and easier to use compared to the previous WeBWorK. He asked us if he had to know how to use LaTeX to write certain expressions; we explained to him that he did need to know it. He also suggested that it would be better if we could make it applicable to randomize the numbers in the same homework sets so that students would be less tempted to copy answers from someone else, enhancing their learning. In general, he thought the code would be easier to write compared to what was required to create his own problem sets in previous WebWorK.

Professor Petruccelli was the last professor who tested the GUI; he also tested the static and the video tutorials. Out of all the professors we tested, he was the most familiar with WeBWorK. He had created problems in WeBWorK before and provided generous feedback on the GUI and tutorials.

Since he utilized the tutorials we created, we did not help him much with the problem creation process. We observed that his WeBWorK file manager was slightly different from the others'. His file manager had a “Choose File” button instead of the “Browse” button. From this test, we also found out that the “Show students partial correct answers” feature for multiple choice problems with multiple answers (e.g. “Choose all that apply” problems) did not work as we expected. It only stated that a sequence of answers was wrong but did not specify which of the answers were wrong.

The professor suggested:

- changing “Re-editing Problems” to “Re-edit Problems” on the Home Page of our website to match grammatically with the rest of the phrases in that bulleted list
- emphasizing that problem authors should not use LaTeX in the answers
- correcting the mistake on the “Open Response with WeBWorK LaTeX” page of our static tutorial, “\(\ LaTeX Code \)” should be “\(LaTeX Code \)” (Quotation marks before the black-slash open parenthesis and after the back-slash closed parenthesis are not included when actually entering in LaTeX in our GUI.)

These professors tested different versions of the WbWrkGUI and the tutorials. The first two professors, whom we tested, did not test the re-editing feature because it was still in development. After each test with a professor, we modified the GUI and/or tutorials according to the professor's suggestion(s). We also added another video tutorial for the re-editing feature; that video has not been officially tested. Originally we did not plan on editing the video tutorials because from the first two testing sessions, both professors stated that they were not interested in video tutorials; however, the last two professors did express interest in the video tutorials.

5. Conclusions and Recommendations

In this chapter, we draw conclusions and make recommendations based on our results.

5.1. Survey and Testing

Since we received little feedback for our online survey, we were not able to draw any precise conclusions. We strongly suggest that future teams aim for a larger sample size.

According to the feedback we had from testing, we conclude that our application is useful to these four professors. Since we only tested four professors in the Mathematical Sciences Department, we do not have enough feedback for analysis. From surveying and testing, we cannot conclude where to improve our application and whether our application will be useful to all problem authors.

5.2. Tutorials

All four professors who have tested the WbWrKGUI website stated that the static tutorials will be very beneficial when he or she would want to use the WbWrKGUI website in the future. Two of the four professors expressed interest in the video tutorials.

Although we used Google Sites, we recommend future IQP teams, who want to work on improving the WbWrKGUI website, to create a website using Adobe Dreamweaver or some other software or website builder because of the limitations of Google Sites. We also suggest creating some type of user guide using Adobe FrameMaker or Adobe RoboHelp. FrameMaker is good for making actual book user guides. RoboHelp makes the user guides that pop up when someone clicks the Help section on a website or software program.

As for the video tutorials, we recommend making tutorials that explain how to make problems requiring differing levels of PG code complexity. These problems will showcase basic problems for the beginner user who just wants easy problems and explain how to create difficult problems with LaTeX syntax, multiple answers, and other features an advanced user might want to use. We also did not explain how to add multiple questions and answers in the “Selecting and Creating Problems” video tutorial because the example we used had only one answer. The explanation and example for that was added in the “Re-editing” tutorial. Having different tutorials based on the complexity of problems, will be less confusing for the video tutorial user to find a tutorial that explains what he or she wants. Although we created video tutorials, we suggest that future teams consider the cost-benefit analysis of making videos because video tutorials take a while to make and are not as easy to edit as a static tutorial. As for software, we recommend what we used, Camtasia and Adobe Premiere.

5.3. Recommendations for Future Projects

During the testing with professors, we found a need for allowing problems to contain random numbers. We did not include this feature because we were afraid that although this is possible on the logic side, it might confuse the user when integrated with the GUI. In addition, many professors are not familiar with LaTeX, so it would be useful for the application to have a built-in LaTeX expression builder.

As mentioned in the Methodology section, the way we are importing .pg files is imperfect. The user is limited to importing only files generated by our application. Another future task would be to improve this method so that it imports other files such as the files from the National Problem Library.

Finally, this standalone java application can be extended to a web-based java applet. A future team could work on making the applet interact with the WeBWorK system directly, which will reduce the effort for professors to upload .pg files.

References

AIM07/Working Groups. (2008, June 17). Retrieved December 8, 2011 from the WeBWorK Wiki: http://webwork.maa.org/wiki/AIM07/Working_Groups

AMS Homework Software Survey. Retrieved from http://www.ams.org/profession/leaders/WATSreport.pdf

Al-Qahtani, S. S., Arif, R., Guzman, L. F., Pietrzynski, P., & Tevoedjre, A. (2010). *Comparing selected criteria of programming languages java, PHP, C++, perl, haskell, AspectJ, ruby, COBOL, bash scripts and scheme revision 1.0.*Cornell University. Retrieved from http://arxiv.org/ftp/arxiv/papers/1008/1008.3434.pdf

Alspach, D. (2008, April 23). Comparison of Maple TA and WebAssign. Retrieved from http://www.math.okstate.edu/mapleta_vs_webassign

Bart. (2007, January 31). How to Produce a Good Video Tutorial. Retrieved from http://www.blendernation.com/2007/01/31/how-to-produce-a-good-video-tutorial/

Bartlett, J. (2005). The art of metaprogramming, Part 1: Introduction to metaprogramming. Retrieved from http://www.ibm.com/developerworks/linux/ library/l-metaprog1/index.html

Bartlett, J. (2006). The art of metaprogramming, Part 2: Metaprogramming using Scheme. Retrieved from http://www.ibm.com/developerworks/linux/library/l-metaprog2/index.html

Bikram. (2011, May 11). Tips for Making a Good Video Tutorial. Retrieved from http://www.itlist.com/tips-for-making-a-good-video-tutorial/

Blackboard Integration. (n.d.). Retrieved November 16, 2011 from http://www.webassign.net/user_support/admin/blackboard_integration.html

Blackwell, A. F. (2001, October). See What You Need: Helping End-users to Build Abstractions. *Journal of Visual Languages & Computing, 12*(5), 475-499. doi:10.1006/jvlc.2001.0216

Bonham, S., Beichner, R., & Deardorff, D. (2001). Online Homework: Does it make a Difference?. *The Physics Teacher, 39*, 293-296. Retrieved from http://www.ncsu. edu/per/Articles/OnlineHomeworkArticle.pdf

Bressoud, D. M. (2009, April). WeBWorK. *Mathematical Association of America*. Retrieved from http://www.maa.org/columns/launchings/launchings_04_09.html

Cervone, D., Gage, M., & Pizer, A. (2007a). Enhancing the problem authoring capabilities of WeBWorK. Retrieved from http://www.aimath.org/pastworkshops/ webwork.html

Cervone, D., Gage, M., & Pizer, A. (2007b). Enhancing the Problem Authoring Capabilities of WeBWorK. Retrieved from http://www.aimath.org/pastworkshops/webworkrep.pdf

Comparing Python to Other Languages. Retrieved from www.python.org/doc/essays/ comparisons.html

Comparison of Homework Functionality. (n.d.). Retrieved November 16, 2011 from http://www.lon-capa.org/comparison.html

Conducting Survey Research, Version 2.0. (1999, March 31). Retrieved from http://www. esf-agentschap.be/uploadedFiles/Voor_ESF_promotoren/Zelfevaluatie_ESF-project/niet%20experimenteel%20onderzoek.pdf

Cozens, S. (2000, February 23). Ten Perl Myths. Retrieved from http://www.perl.com/pub/2000/01/10PerlMyths.html

Create Your Own Questions. (n.d.). Retrieved November 16, 2011 from http://www.webassign.net/features/create_your_own_questions.html

DarleneVictoria. (2009, October 6). How to Create Effective Video Tutorials [Video file]. Retrieved from http://www.youtube.com/watch?v=uVdWukvT0Ew

Displaying Notation with WaTeX. (2011, October). Retrieved November 16, 2011 from http://www.webassign.net/manual/instructor_guide/c_i_displaying_notation_watex. htm

Dreibelbis, J. D. (n.d.*). WeBWorK—A Free Online Homework System* [PDF document]. Retrieved from http://people.rit.edu/jdd5747/webwork.pdf

Enhancing the problem authoring capabilities of WeBWorK. (2007, July 30). Retrieved from http://webcache.googleusercontent.com/search?q=cache:_3si4ohjw0sJ:www. aimath.org/WWN/webwork/webwork.ps+webwork+pg+language+problem+ authoring+troubles&cd=17&hl=en&ct=clnk&gl=us

Fiore, T. M., & Agarwal, M. (2011, January 26). *Implementation of WeBWork in 104-227 and Team-Built Assignments: Pros and Cons.* Retrieved from http://www-personal.umd.umich.edu/~tmfiore/1/FioreWeBWorkTalk.pdf

For Blackboard Software. (n.d.). Retrieved November 16, 2011 from http://www.maplesoft.com/products/mapleta/blackboard.aspx

Gok, Tolga. (2011, August 4). Comparison of student performance using web- and paper- based homework in large enrollment introductory physics courses. *International Journal of the Physical Sciences, 6*(15). 3740-3746. Retrieved from http://www.academicjournals.org/IJPS/PDF/pdf2011/4Aug/Gok.pdf

Gotel, O., Kline, R., Scharff, C., Wildenberg, A., Baldwin, J., Crupi, E., & Estrellado, T. (2006 March 3). Retrieved from http://www.csis.pace.edu/~scharff/webwork/ WWtutorial.htm

GUI Definition. Retrieved from http://www.linfo.org/gui.html

Hartley, C., Schendel, E., & Neal, M.R. (1999). Writing (online) *spaces*: composing Webware in Perl. *Computers and Composition, 16*(3), 359-370. doi:10.1016/S8755-4615(99)00016-X

Heck, A. (2004, November 2). Assessment with Maple T.A.: Creation of Test Items. Retrieved from http://www.fi.adeptnordic.com/products/mathsim/mapleta/ MapleTA_whitepaper.pdf

Heffernan, C., & O'Connor, C. (n.d.). ASSISTment: a teacher uses the internet to collect, share and respond to data from her students. *NCTM Mathematics Teacher in the Middle School Journal*. Retrieved from http://www.grantsandresearchhelp.us/ research/ASSISTment-Description.pdf

Help/build/sequence/index. (2011, November 2). Retrieved November 16, 2011 from the teacherwiki.assitment.org Wiki: http://teacherwiki.assistment.org/wiki/Help/build/ sequence/index

How to use Math functions. (2011, October 24). Retrieved November 16, 2011 from the teacherwiki.assitment.org Wiki: http://teacherwiki.assistment.org/wiki/How_to_ use_Math_functions

Introduction. (2011, September 7). Retrieved November 11, 2011 from the WeBWorK Wiki: http://webwork.maa.org/wiki/Introduction

Li, Z., & Xia, Y. (2011). *WebWork Problems for Statistics I.* Worcester, MA: Worcester Polytechnic Institute.

Jones, C. G. (2008). *Student Perceptions of the Impact of Web-Based Homework on Course Interaction in Introductory Accounting.* Issues in Information Systems, 9(1), 223-230. Retrieved from http://www.iacis.org/iis/pdf/S2008_1116.pdf

Libraries. (2011, January 25). Retrieved November 16, 2011 from the teacherwiki.assitment.org Wiki: http://teacherwiki.assistment.org/wiki/Libraries

LON-CAPA. (n.d.). Retrieved from http://www.lon-capa.org/papers/flyer_for_web.pdf

Lucas, A. R. (n.d.). Using WeBWorK, a Web-Based Homework Delivery and Grading System, to Help Prepare Students for Peer Instruction. Retrieved from http://www.dillgroup.ucsf.edu/~alucas/webworks-PRIMUS.pdf

MacVittie, D. (1999). *Contender or champ? judging java*. *Network Computing, 10*(13), 84-94. Retrieved from http://au4sb9ax7m.search.serialssolutions.com/?ctx_ver=Z39.88-2004&ctx_enc=info%3Aofi%2Fenc%3AUTF-8&rfr_id=info:sid/summon.serialssolutions.com&rft_val_fmt=info:ofi/fmt:kev:mtx:journal&rft.genre=article&rft.atitle=Contender+or+champ%3F+Judging+Java&rft.jtitle=Network+Computing&rft.au=Don+MacVittie&rft.date=1999-06-28&rft.issn=1046-4468&rft.volume=10&rft.Issue=13&rft.spage=84&rft.externalDBID=NETC&rft.externalDocID=42790645

Main Page. (2011, September 7). Retrieved November 11, 2011 from the WeBWorK Wiki: http://webwork.maa.org/wiki/Main_Page

MathFlow. (n.d.). Retrieved from http://www.dessci.com/en/products/mathflow/default.htm

Mendicino, M., Razzaq, L., and Heffernan, N. T. (2009) *A Comparison of Traditional Homework to Computer-Supported Homework*, JRTE, 41(3), 331–358, Retrieved from http://teacherwiki.assistment.org/wiki/images/6/6b/Mendicino.pdf

Mischook, S. (2007). How to Create Video Tutorials. Retrieved from http://www.killersites.com/blog/wp-content/uploads/2007/05/idea22_how-to-create-learning-video.pdf

National Problem Library. (2011, September 7). Retrieved November 11, 2011 from the WeBWorK Wiki: http://webwork.maa.org/wiki/National_Problem_Library

Nisley, E. (2002). *Java: Stirring the cup*. *Dr. Dobb's Journal, 27*(2), 90-93. Retrieved from http://au4sb9ax7m.search.serialssolutions.com/?ctx_ver=Z39.88-2004&ctx_enc=info%3Aofi%2Fenc%3AUTF-8&rfr_id=info:sid/summon.serialssolutions.com&rft_val_fmt=info:ofi/fmt:kev:mtx:journal&rft.genre=article&rft.atitle=Java%3A+Stirring+the+cup&rft.jtitle=DR+DOBBS+JOURNAL&rft.au=Nisley%2C+E&rft.date=2002-02-28&rft.pub=MILLER+FREEMAN%2C+INC&rft.issn=1044-789X&rft.volume=27&rft.issue=2&rft.spage=90&rft.epage=90&rft.externalDBID=IDRD&rft.externalDocID=98792050

Oud, J. (2009). Guidelines for effective online instruction using multimedia screencasts. *Reference Services Review, 37*(2), 164-177. doi: 10.1108/00907320910957206

Pascarella, A. M. (2004). The influence of web-based homework on quantitative problem solving in a university physics class. NARST 2004 annual meeting.

PREP 2011 Notes June 25. (2011, June 26). Retrieved December 8, 2011 from the WeBWorK Wiki: http://webwork.maa.org/wiki/PREP_2011_Notes_June_25#Davide

Question Options. (n.d.). Retrieved November 16, 2011 from http://www.webassign.net/features/question_options.html

Rajaraman, J. (1998). *Programming languages: A brief review*. *Resonance, 3*(12), 43-54. Retrieved from http://www.springerlink.com/content/7w2t4j38n4128120/ fulltext.pdf

Release notes for WeBWorK 2.4.9. (2011, May 15). Retrieved December 9, 2011 from the WeBWorK Wiki: http://webwork.maa.org/wiki/Release_notes_for_WeBWorK_ 2.4.9

Richards-Babb, M., Drelick, J., Henry, Z., & Robertson-Honecker, J. (2011). Online Homework, Help or Hindrance? What Students Think and How They Perfom. *Journal of College Science Teaching, 40*(4), 82-93. Retrieved from http://www.cideronline.org/podcasts/pdf/9.pdf

Sheppard, D. (2000, October 16). Beginner's Introduction to Perl. Retrieved from http://www.perl.com/pub/2000/10/begperl1.html

Smart Survey Design. Retrieved from http://s3.amazonaws.com/SurveyMonkeyFiles/ SmartSurvey.pdf

Survey Design Tutorial. Retrieved from http://www.statpac.com/surveys/

System Requirements. (n.d.). Retrieved November 16, 2011 from http://www. maplesoft.com/products/system_requirements.aspx#mapleta

Tip 20: Six Steps to Effective Tutorials. (n.d.). Retrieved from http://kineo.com/elearning-tips/tip-20-six-steps-to-effective-tutorials.html

Ward, M. (n.d.). The Anatomy of a Great Tutorial. Retrieved from http://spyrestudios.com/the-anatomy-of-a-great-tutorial/

WebAssign creating questions guide. (2011). USA: Advanced Instructional Systems, Inc. Retrieved from https://www.webassign.net/manual/Question_Coding_Reference.pdf

WebEQ Developers Suite Reaches End of Life. (n.d.). Retrieved November 16, 2011 from http://www.dessci.com/en/support/webeq/eol.htm

What is Maple? (n.d). Retrieved December 6, 2011 from http://www.maplesoft.com/products/Maple/features/index.aspx

What is Python good for. Retrieved from http://docs.python.org/faq/general.html#what-is-python-good-for

What is WeBWorK? (2011). Retrieved December 6, 2011, from http://webwork.maa.org/intro.html

Why Maple T.A.? (n.d.). Retrieved November 16, 2011 from http://www.maplesoft.com/products/mapleta/why.aspx

Ziemer, W. K. (n.d.). WeBWorK: An Open-Source Online Homework System. Web-Enabled Learning Environment, 0, 169-171. Retrieved from http://www.aaas.org/publications/books_reports/CCLI/PDFs/06_WLE_Ziemer.pdf

Appendix

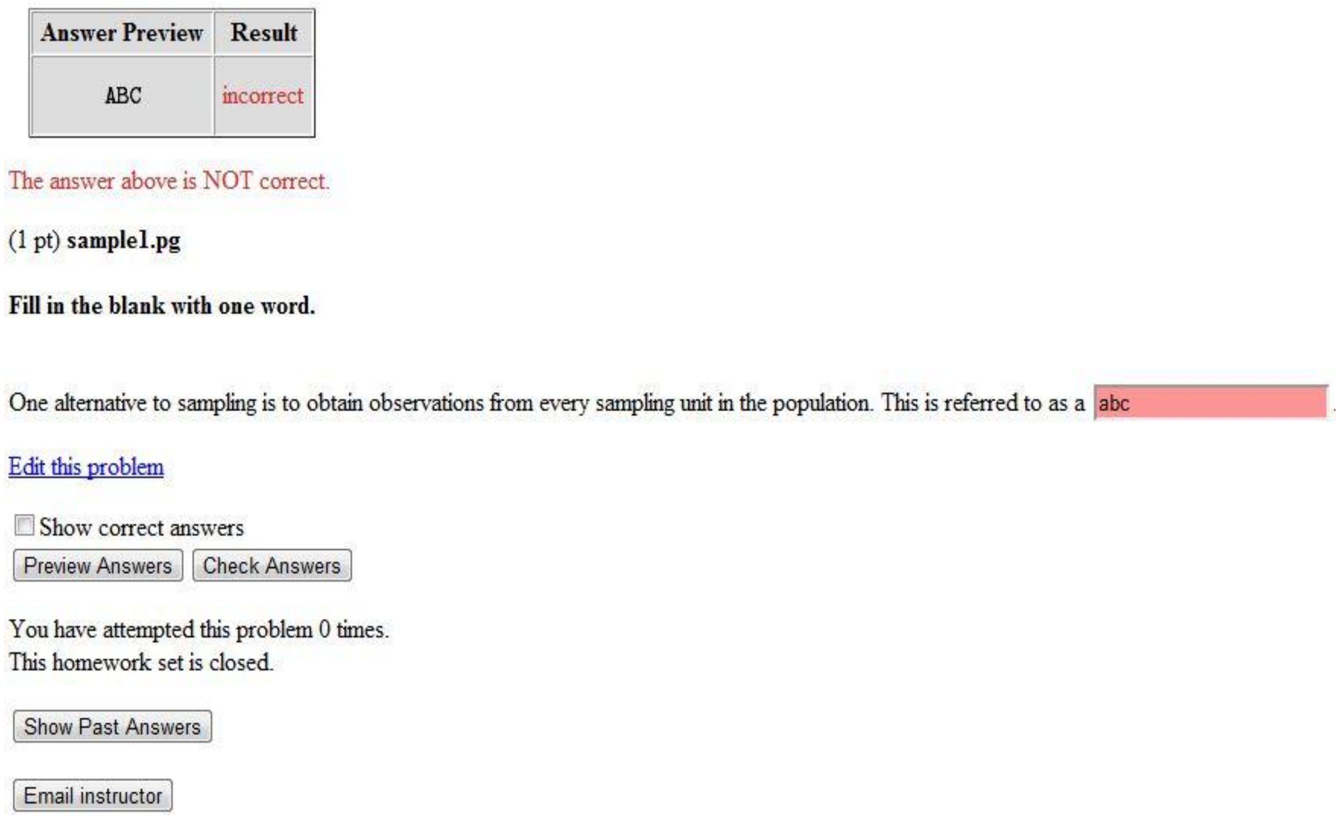
Appendix A: PG Code and Examples of Problems

```
DOCUMENT();        # This should be the first executable line in the problem.
loadMacros("PGbasicmacros.pl","PGchoicemacros.pl", "PGanswermacros.pl");
TEXT(beginproblem(), $BR,$BBOLD, "Fill in the blank with one word.", $EBOLD,
$BR,$BR);                                          The Header

$showPartialCorrectAnswers = 0;

        BEGIN_TEXT
        $PAR One alternative to sampling is to obtain observations from every sampling
unit in the population. This is referred to as a  \{ ans_rule(20) \}.       ← The Problem
        END_TEXT
        ANS( str_cmp( "census" ) );        ← The Answer.

ENDDOCUMENT();
```



Answer Preview	Result
ABC	incorrect

The answer above is NOT correct.

(1 pt) **sample1.pg**

Fill in the blank with one word.

One alternative to sampling is to obtain observations from every sampling unit in the population. This is referred to as a abc .

Edit this problem

☐ Show correct answers

Preview Answers Check Answers

You have attempted this problem 0 times.
This homework set is closed.

Show Past Answers

Email instructor

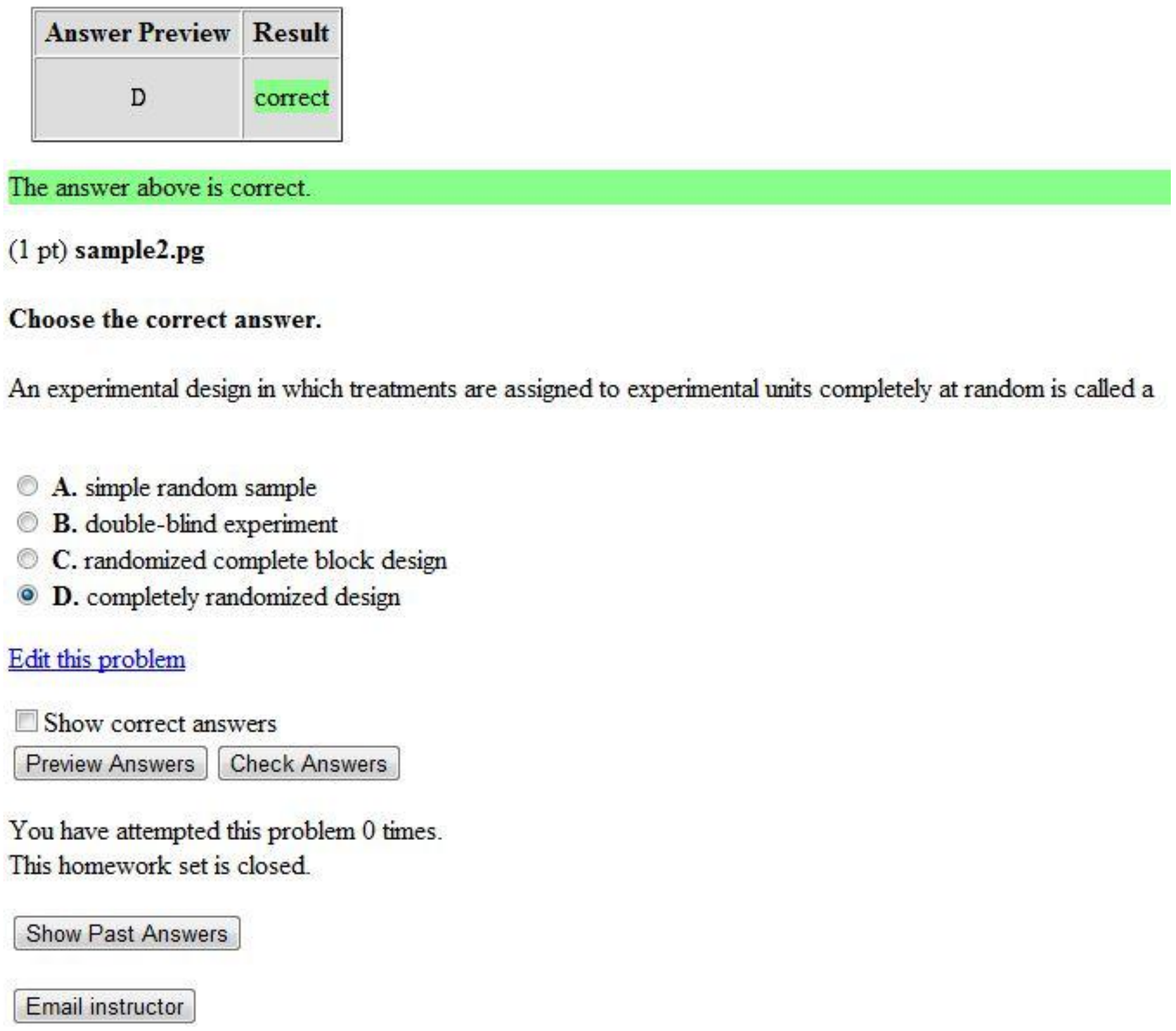
Figure 8: Example of the PG code and Final WeBWorK View of a Fill-In the Blank Problem

```
DOCUMENT();        # This should be the first executable line in the problem.
loadMacros("PGbasicmacros.pl","PGchoicemacros.pl", "PGanswermacros.pl");
TEXT(beginproblem(), $BR,$BBOLD, "Choose the correct answer.", $EBOLD, $BR,$BR);
                                                   Header
$showPartialCorrectAnswers = 0;
# Make a new multiple choice object.
$cmc = new_multiple_choice();

$cmc->qa(
                  'An experimental design in which treatments are assigned to experimental
units completely at random is called a', ← The Problem
          'completely randomized design' ← The Answer
          );
$cmc ->extra("randomized complete block design","double-blind experiment","simple
random sample"); ← Other choices

BEGIN_TEXT

\{$cmc -> print_q \}
$PAR
\{$cmc -> print_a\}
END_TEXT
# Enter the correct answers to be checked against the answers to the students.
ANS(radio_cmp($cmc->correct_ans));
ENDDOCUMENT();
```



Answer Preview	Result
D	correct

The answer above is correct.

(1 pt) **sample2.pg**

Choose the correct answer.

An experimental design in which treatments are assigned to experimental units completely at random is called a

- **A.** simple random sample
- **B.** double-blind experiment
- **C.** randomized complete block design
- [✓ **D.** completely randomized design]

You have attempted this problem 0 times.
This homework set is closed.

Figure 9: Example of the PG Code and Final WeBWorK View of a Multiple Choice Problem