AutoEd

An Online Assignment Generation and Marking System

By

Alyosha Pushak

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science Honours

In

The Irving K. Barber School of Arts and Sciences

(Honours Computer Science Major Computer Science)

The University of British Columbia

(Okanagan)

April 2011

Abstract

AutoEd is an open source web-based system that allows professors to give students assignments made of randomly generated questions. The system was developed to improve student learning and provide students with easy access to a potentially unlimited amount of practice questions in a broad range of subjects, as well as saving time for professors and TAs by auto-marking assignments. Professors create question templates from which random questions are generated. Questions are written in HTML with special <eqn> tags that specify calculations and random variable assignment. Numerical values and units can be randomized. Students and professors can search for question templates by category, and then try practice questions generated from those templates. AutoEd provides immediate feedback to students, and can provide hints as to what they did wrong and how to solve the question. The server code, written in Java, handles the question generation and marking. The templates and users' tests, answers and marks are stored in a database.

Table of Contents

Table of Figures

1 Introduction

1.1 Motivation

Assignment preparation and marking, especially for first year classes with over a hundred students, is very time consuming. Automating this process can free up instructors' and TAs' time for more valuable student contact. Online test systems can also be a resource for students to help them learn, and provide more standardized evaluations of students and instructors. While many such systems exist already, most are not free. Either the school or the students have to pay for them. Many only allow static questions, so there is no value in repeating the questions. We decided to build our own system rather than use a pre-existing system.

1.2 Thesis Statement and Contributions

The contribution of this thesis is the development and evaluation of the AutoEd assignment system. The thesis is that the system will improve student performance and engagement in large classes.

AutoEd is a web-based system for automated assignment generation and marking. Instructors create question templates from which random questions are generated. It is also a tool to help students learn. As some values are randomized, students can repeat a question with different values until they feel confident they know how to solve it. The system provides students with immediate feedback. The goal is to create a resource for students providing easy access to a potentially unlimited amount of practice questions in a broad range of subjects. The system was evaluated by a first year physics class during first term. Over second term, two more sections of the first year physics course used the system, and a third year pharmacology class used it for one assignment.

1.3 Definitions

Users are divided into two categories: **instructors** and **students**. Students can search for and practice questions, and take assignments in the **courses** they are registered in. Instructors can do everything students can do. Additionally they can create question templates, and assign tests in any courses they instruct. They can view the assignments and marks of any students in their courses.

Question templates are used to generate random questions. Variables in the question template are assigned random values, resulting in a **question**. Question Templates are made of one or more **question template parts**. For instance, a multi-part question with parts A) B) and C) would have three question template parts. During question generation, each question template

part generates a **question part**. Questions and question parts can store a user's answers and marks. **Test templates** are made up of multiple question templates, and are used to generate **tests**. Tests belong to a user, and can store that user's overall mark.

1.4 System Overview

The users' test scores and answers, the templates for questions and tests, and information on users and courses are stored in a Microsoft SQL Server database on a campus server, cssql.ok.ubc.ca. The server code, written in Java, runs on another on-campus server, cs-suse-4.ok.ubc.ca. It handles the question generation and marking, and interacts with the database. The front-end pages running client side are written in HTML and JavaScript, and send Ajax requests to the server to mark questions and perform various other tasks without reloading the page. Users interact with the system using a web browser. The system was tested and works with Internet Explorer (Version 7+), Mozilla Firefox and Google Chrome.

2 Background

The idea of question templates is not new. One of the first systems produced was WWWAssign built by Dr. Larry Martin for physics courses. Initially freely released, this system is no longer supported and has been replaced by the commercial product WebAssign [6]. WebAssign has content and questions from multiple textbook publishers.

There are other similar commercial systems most often affiliated with textbook publishers including Aplia [7] (mostly for social sciences), Mastering Physics [8], Gradiance [9] (computer science), and Carnege Learning (mathematics) [10]. The advantages of using these products are that the questions are completed and verified, there may be questions associated with the course textbook, and the web hosting is provided by the companies so no technical setup is required. The disadvantages are cost and inflexibility. The cost of these systems ranges from approximately $15 to $100 per student per course. Given the large enrolments for classes that typically use these systems, this is a cost most likely placed on the student rather than the institution. Although both parties see a benefit (students for learning, the institution for reduced costs), it is primarily supported by students, who are resistant to increased costs. Further, it is challenging to create new questions in these systems and their costs force instructors to use them significantly in a course. It is not cost-effective to use them for a few questions that require specific repetitive practice or produce your own questions for use.

Another alternative is that most course management systems have built-in support for quizzes and tests including some such as WebCT Vista [11] that have limited support for randomized questions. If the institution has a course management system deployed, then simple questions can be done this way. We evaluated the support for randomized questions in WebCT Vista and

found them reasonable for basic questions but not acceptable for more detailed formula-based questions.

Finally, there have been several university led efforts on building these systems and providing access. One such system is the Quest Service [12] at The University of Texas at Austin. This system has questions for many sciences and outside institutions can get access by requesting UT Austin accounts.

LON-CAPA [13] is an open source system built primarily at Michigan State University. It has hundreds of thousands of resources and questions that are shared by multiple institutions. Institutions are part of a sharing network for distributing questions and resources. However, they do not release questions outside of the institutions using the system, or to students (so students cannot acquire the answers). We wanted to build a system where anyone, including students, could access the questions.

There has been some informal evidence that these systems have improved efficiency, especially for large first-year classes, and student learning and educational outcomes. Students enjoy the immediate response of the system and overall performance on tests and lectures have improved.

3 User Manual

3.1 General View

3.1.1 Logging In

On the main page at cs-suse-4.ok.ubc.ca/autoedu, you must log in before you can continue. Students default username and password are their student number. You can log into a demo student account with user name and password “demo”.

3.1.2 My Courses

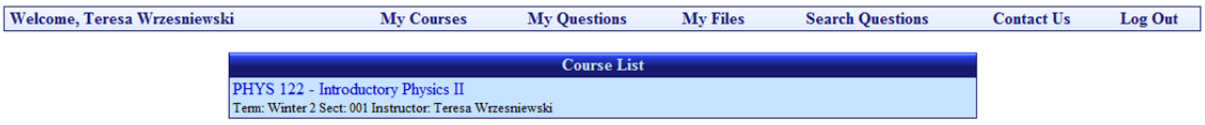

Figure 1: My Courses

Once you log in you will be directed to the My Courses page. It lists the courses you are registered in. Instructors will see the courses they instruct. Course details—the course number,

name, term, section number and instructor name—are shown. Clicking the course number or name will bring you to the course page for that course.

3.1.3 Searching Questions

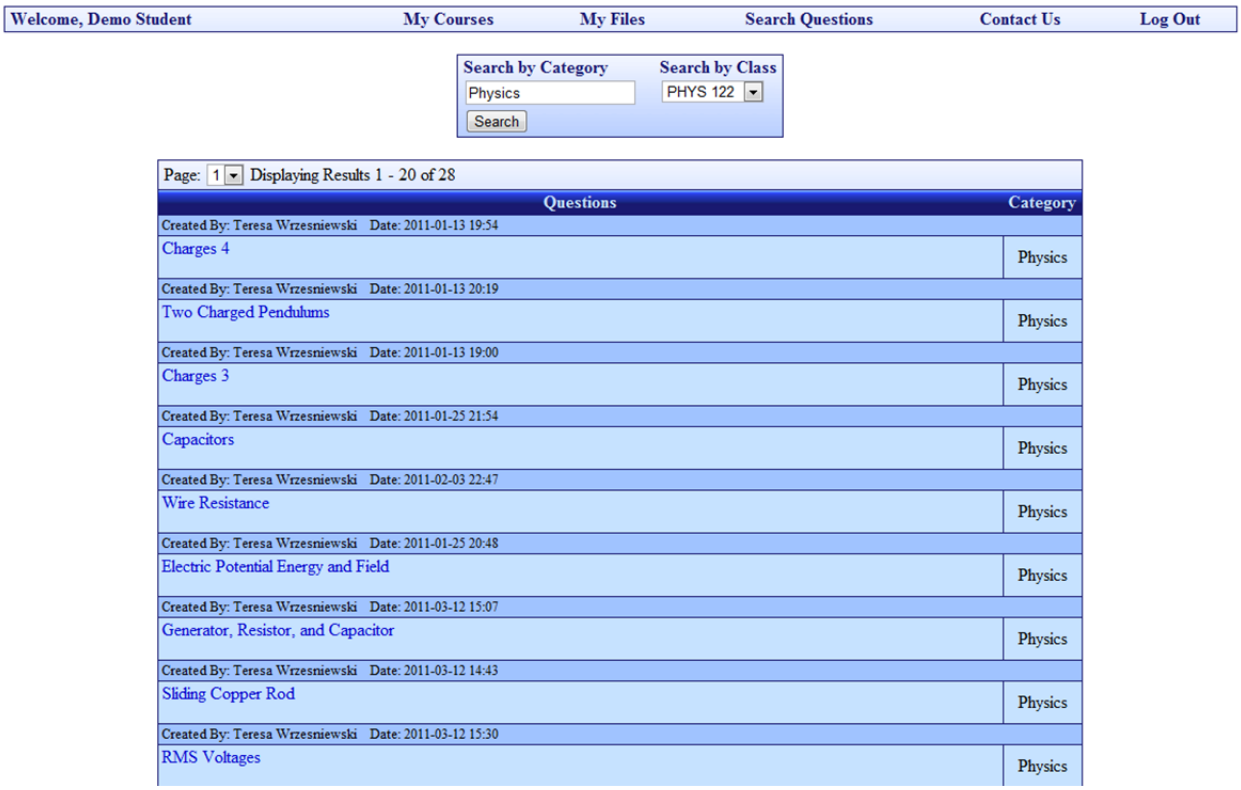

Figure 2: Search Questions

You may search for question templates by category and/or by course. If you leave both category and course blank, the search will retrieve all question templates. Twenty questions are displayed per page. The question template name, description, author's name, creation date, and category are listed for each template. Clicking the question template name allows you to practice a question generated from that template. Instructors will see a "copy" link for each template. Clicking this will bring you to the question editing page, with the data loaded for that template. Clicking save will create a copy of the question that belongs to you, with any changes you made in the editor. Instructors will also see a "modify" link for any template they created, allowing them to edit the template. For more information on creating/editing question templates see sections 3.3.3 and 3.3.4.

3.2 Student View

3.2.1 Course Page

Each course page has a header with three rows. The first row displays the course name and number. Second are links to the assignments page and the questions page. Third is course information: the term, section number, and instructor name.

3.2.2 Assignments Page

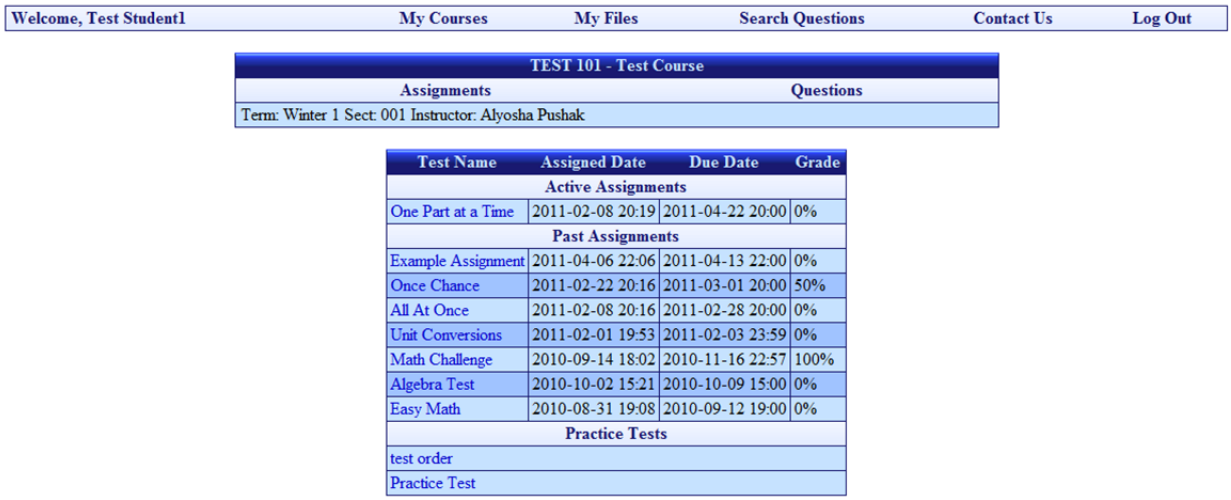

Figure 3: Assignments Page

When you click on a course from “My Courses” you will be brought to the course’s assignments page. A table of assignments and practice tests is shown here, listing active assignments first, then past assignments, and lastly practice tests. Active assignments with nearer due dates appear first. The assignment name, due date, assigned date, and your grade for each assignment is listed. For practice tests only the name of the test is listed.

3.2.3 Practicing Questions

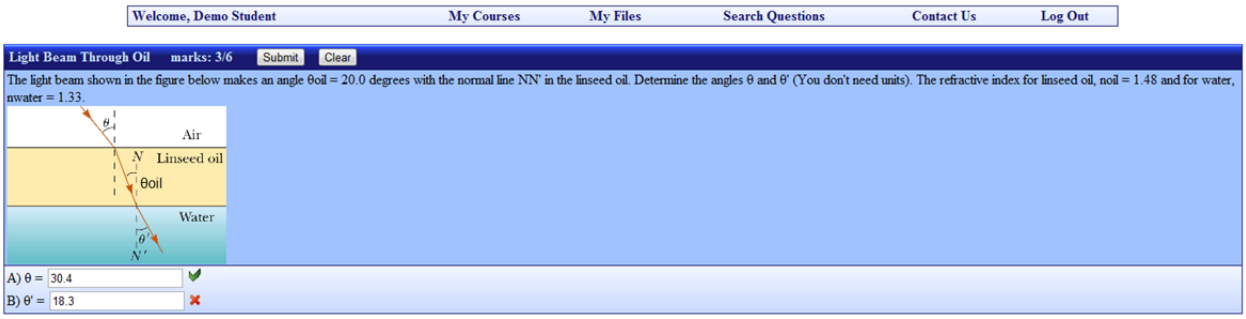

Figure 4: Practice Question

On the questions page within a course you will see all the questions for that course, along with the question's category. Click on a question's name to practice an instance of that question. You may also search for questions to practice from the Search Questions page (see section 3.3.1). When viewing a question you will see a page as shown in Figure 4. Questions have a question header, which shows the question name, your marks and the total marks for the question, and submit and clear buttons. Below is the text for the question. Questions have one or more questions parts to them, each of which requires an answer. When you have answered all the question parts, click the submit button to have your answers graded. You may submit answers as many times as you want. Green check marks will appear next to correct answers, whereas red exes indicate incorrect answers. The clear button will clear your answers to each question part. Refreshing the page will load a new instance of the question with different values. For some incorrect answers a "Show Hint" button will appear. Click the button to view a hint.

3.2.4 Entering Answers

In this section we discuss the format in which answers are entered.

- **Numerical Values –** decimal answers within a given tolerance of the correct answer are accepted. The default tolerance is 2%, though the instructor may set it to 0%, in which case the exact numerical value is needed. For integer values the exact value is usually required, unless the instructor sets up the answer as a decimal answer.
- **Scientific Notation –** You may use scientific notation of the form 6.1E-5.
- **Units –** Some answers require units. Units follow the numerical value, which may or may not be separated by a space. Units are case sensitive. If you forget to include units for a question that requires them, you will be given a hint reminding you to include units. If you provide the correct answer but the wrong units, you will also be given a hint informing you so. The marker can convert unit prefixes; for example, if the expected

answer is 1.1 m, the answer 110 cm would be marked correct. The instructor may disable unit conversions for a given question, in which case the instructor should tell you in the question which units the answer expects. For a full list of acceptable unit prefixes see section 3.3.4.5.

There are three ways a question part may expect answers inputted, as follows:

- **Text –** A text box is provided for you to type the answer.
- **Multiple Choice –** A list of possible answers is provided. Click the radio button next to the answer you think is correct.
- **Text with Solver –** Like the text option, though this time a "solve" button appears next to the text box. You may enter simple equations into the text editor, beginning with the '=' character. Equations can use the following operators: + (addition), - (subtraction), / (division), * (multiplication) and ^ (exponents). You may use parenthesis to specify order of operation. Clicking the solve button evaluates the expression you entered. Expressions will also be evaluated when you click the submit button to have your answer graded.

3.2.5 Practicing Tests

Figure 5: Practice Test

From the course assignments page, click on a practice test to practice the test. You may also practice assignments. Clicking the assignment name will list your attempts at that assignment.

Click “Practice this Test” to practice the test. You may practice a test as many times as you like. At the top of the page you will see the test header. This header will be the same when doing an assignment. On the left the question numbers are listed along with you mark and the total marks for each question, and the test total and your overall mark and percentage. Clicking a question number will scroll the page to that question. On the right the time you’ve had the page opened is listed. In the center a custom message from the instructor may appear, if the instructor creates one. Below the header the questions appear as they did when practicing a question. Inputting and submitting answers is done the same as it is for practicing questions.

3.2.6 Assignments

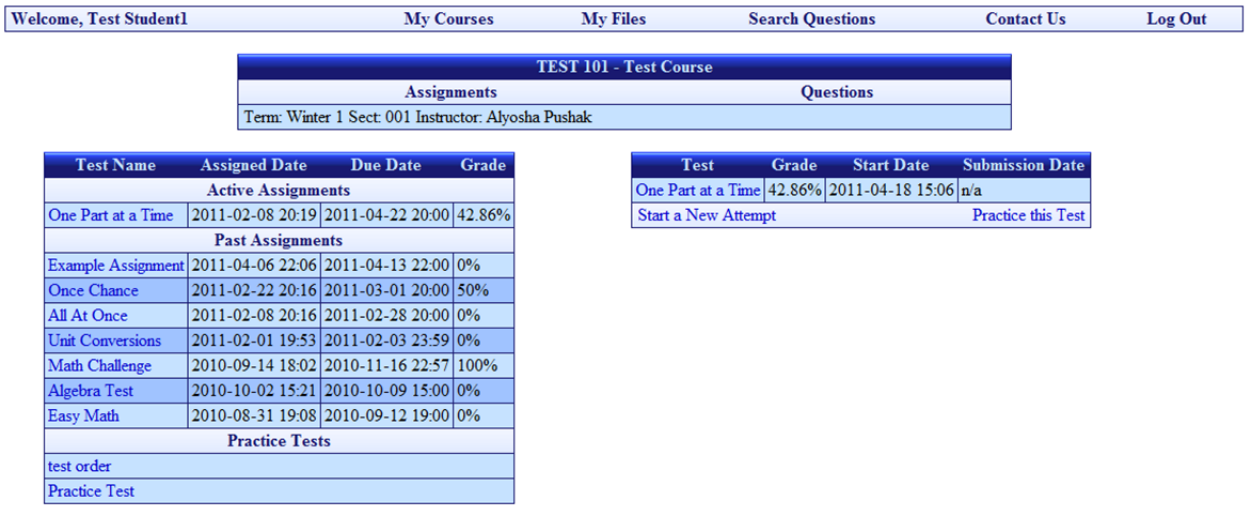

Figure 6: Assignment Attempts

Figure 7: Assignment

On the course assignments page, click an assignment to view your attempts for that assignment, along with your grade, start time, and submission time for each attempt. Each attempt has its own set of questions and saves your answers and marks. Only your attempt with the highest grade will count. Click “Start a New Attempt” to begin an attempt. Any answers you submit that are correct will be saved. You can leave the page and your answers will be there when you return. To resume an attempt where you left off, click on the attempt in the list of attempts on the course assignment page. You may not start new attempts once the due date is passed. If you open an attempt after the due date, you may view your attempt, but you won’t be able to change your answers. There are different types of assignments as follows:

- **All at Once –** Like when practicing questions and tests, all parts of a question are marked at once. Once you submit your answers you may not change them, however, you may retry another instance of the question with different random values as many times as you like, by clicking “retry” in the question header. The number of times you try each question is listed in the question header. This is the only type of assignment with a retry button.
- **One at a Time –** You submit answers to one question part at a time. In a question with multiple parts, you must answer the previous part correctly before moving on to the next part. You may submit answers as many times as you like. The inputs for the question parts you cannot answer are greyed-out.
- **One Chance –** Only one attempt is allowed, and you may only submit your answers once. All parts of a question are marked together. Be sure to answer each question part before clicking “submit” for that question.

Once you’ve answered all of the questions, you may submit your assignment by clicking “submit” on the right side of the assignment header. You will still get your grade if you don’t click submit. The assignment submission is to give the instructor an indication of how many students completed the assignment, and when.

3.3 Instructor View

3.3.1 Course Page

The course page view for instructors is similar to the student view. The differences are detailed here.

In the course header there is an additional link to the “Student List,” where the list of students enrolled in the course, and their student numbers can be viewed

3.3.2 Viewing Test Results

Student Grades for Once Chance - View Statistics - Download Results

Student Name	Grade	Start Date	Submission Date
Alyosha Pushak	12.5%	2011-02-22 20:16	n/a
Test Student1	50%	2011-02-22 20:31	2011-02-22 20:32

Figure 8: Student Grades

On the assignments page, below the list of assignments, when an assignment is clicked on, a list of students will appear below the table of assignments, listing the date the student started and completed the assignment (n/a if the student hasn't submitted the assignment), and their grade (percentage) for their best attempt at the assignment. Clicking a student's name allows you to view the student's assignment and see the answers they submitted. Above the list of student marks, the assignment name along with the links "View Statistics" and "Download Results" are listed.

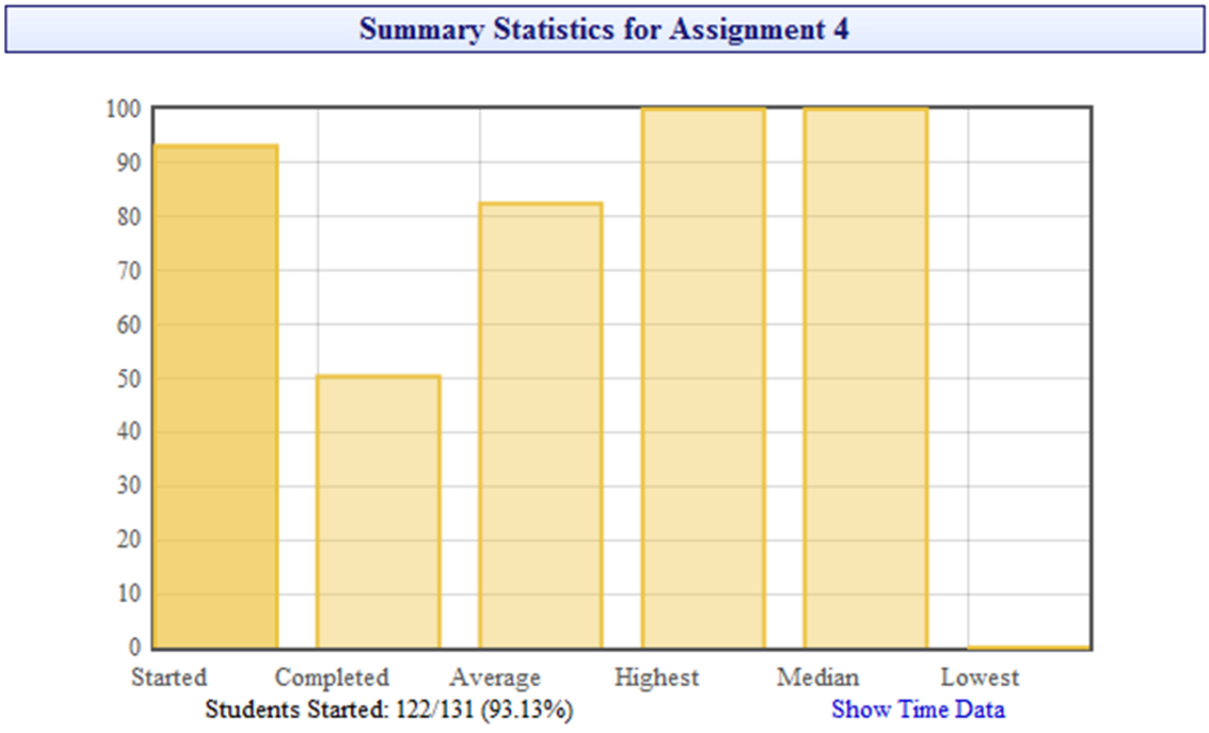

Figure 9A: Assignment Results

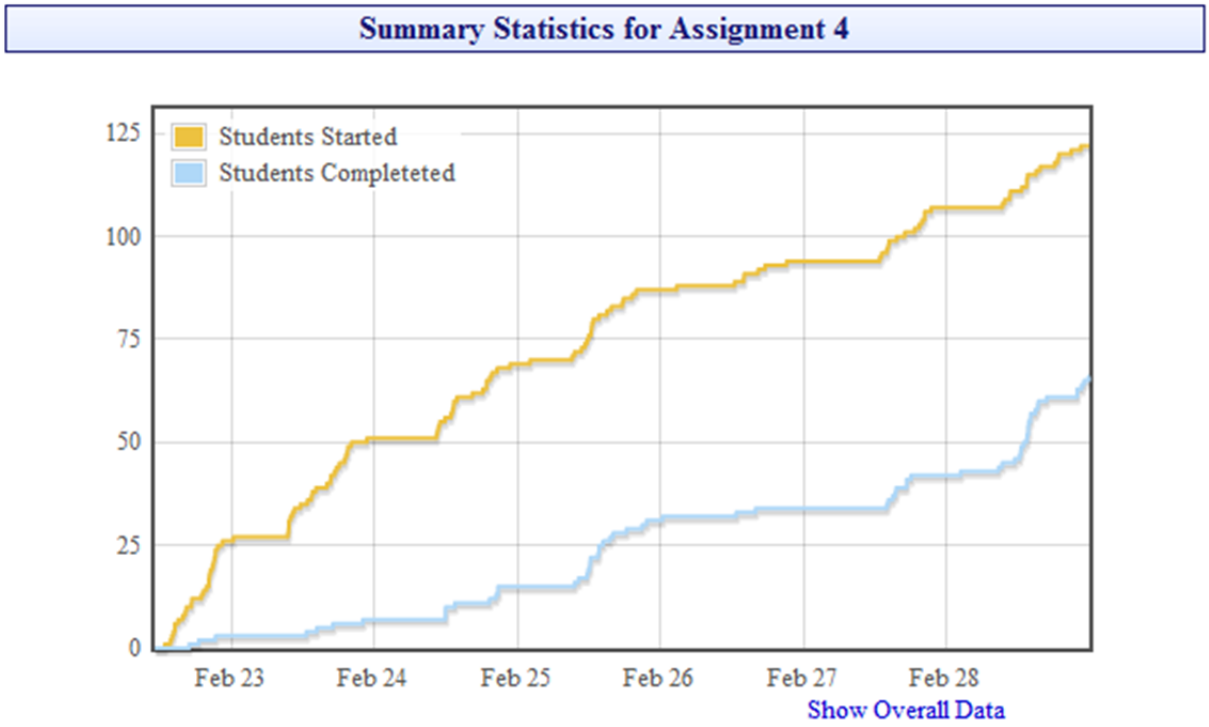

Figure 9B: Assignment Results

Click “View Statistics” to see a graph of the assignment results. You will see a bar graph displaying the number of students who started the test, the number that completed the test, the average, highest mark, median, and lowest mark, as percentages. Mouse over a column to view the percentage and, for students started and completed, the number of students started/completed over the total number of students in the class. Instructors may try assignments. Their scores are not reflected in this data. Click “Show Time Data” to view a graph of students started and students completed over time. The start and end points of the graph are the assigned and due date of the assignment. You can return to the original graph by clicking “Show Overall Data.” On the assignments page, clicking “Download Results” will download a .csv (comma separated values) file containing the student name, number (prefixed with an ‘s’ so as to be compatible with WebCT), mark, and grade (percentage).

3.3.3 Creating a Question Template

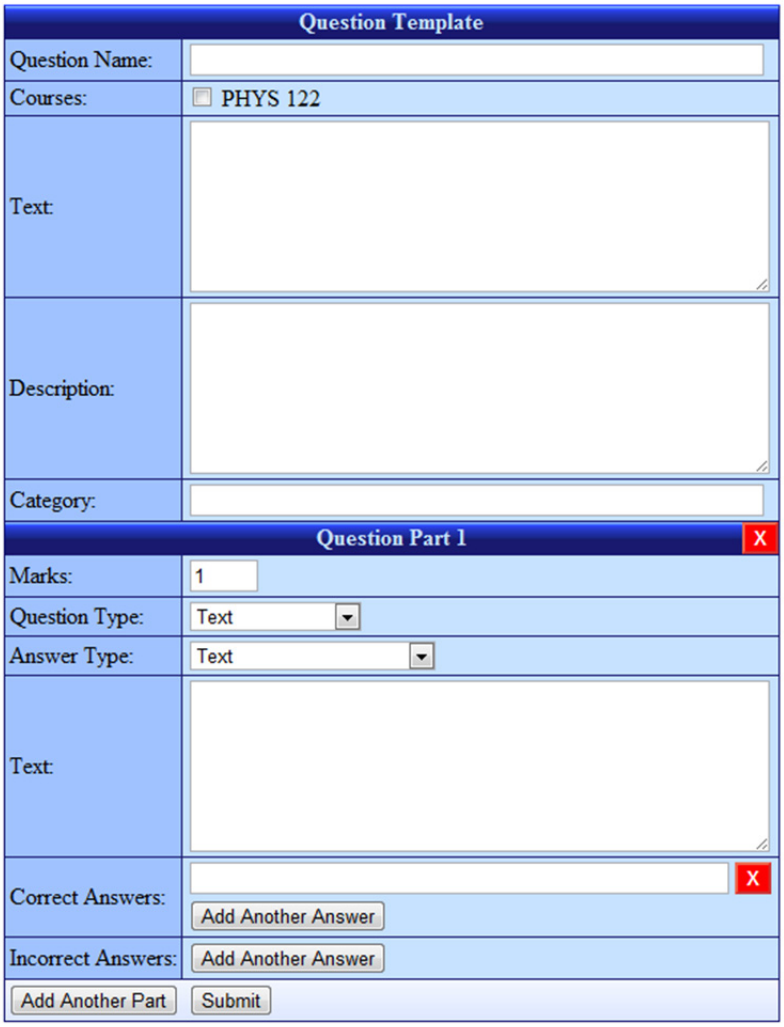

Figure 10: Question Template

On the My Questions page click "Create a Question" to view the question template creation page. You can edit the following fields. Unless otherwise indicated, fields may be left blank.

Question Template Fields

- **Question Name –** The name of the question. Cannot be left blank. Max 100 characters.
- **Courses –** The courses you instruct appear here. Check the courses you want this question to belong to. Students in those courses can then see and practice this question.
- **Text –** The text for this question. Written in HTML with special <eqn> tags to specify calculations and random variable assignment. See the section 3.3.4 for details on the syntax for <eqn> tags.

- **Description –** A description of the question. It appears with the question template in search results.
- **Category –** The category of this question.

Question Template Part Fields

- **Marks –** The amount of marks awarded for a correct answer to this question part. Must be an integer greater than 0.
- **Question Type –** Determines the input used for the answer. Must be one of the following:
    - Text – the answer is entered in a text box.
    - Text with Solver – the answer is entered in a text box. Computations beginning with an "=" can be evaluated when the answer is submitted or when the "solve" button is pressed. For more information on the solver, see page 36 in the appendix.
    - Multiple Choice – The answer is selected from a list of possible answers. Answers entered in the "Incorrect Answers" field will appear as possible answers.
- **Answer Type –** Determines how the answer is interpreted by the marker. Must be one of the following:
    - Text – answer is matched against a string, case insensitive.
    - Text Case Sensitive – answer is matched against a string, case sensitive.
    - Integer – answer is compared with an integer.
    - Integer with Units – answer must have an integer value and units which must match a correct answer. Unit conversions aren't allowed for integers, as the conversions could result in non-integer values.
    - Real Number – answer is a double. It is marked correct if it is within a certain range of the correct answer.
    - Real Number With Units – answer is a double with units.
    - Integer Vector – a list of integers.
    - Integer Vector with Units – a list of integers followed by units.
    - Real Vector – a list of doubles.
    - Real Vector with Units – a list of doubles followed by units.
    - Text Vector – a list of strings, all compared with case insensitive.

Some answer types have additional fields that must be filled out, as follows:

- Precision – Must be filled out for all types containing real numbers. The answer is marked correct if it is in the range of the correct answer + or – correct answer * precision. Must be an integer between 0 and 100 inclusive. Default value is 2.

    - Enable Unit Conversions – this option is available for any types containing real numbers with units. If enabled, if the base units in the answer match the correct answer's base units but the prefixes don't match, the answer will be converted to match the prefix of the correct answer.
- **Text –** The text that appears for this question part. Written in HTML using special <eqn> tags to specify calculations and random variable assignment.
- **Correct Answers –** The correct answers. Use <eqn> tags to calculate the answers. Click "Add Another Answer" and another text box will appear for inputting another answer. At least once correct answer must be given.
- **Incorrect Answer –** Click "Add Another Answer" and two text boxes will appear; one for the answer and one for a hint. The hint will appear when the student tries the corresponding incorrect answer. In multiple choice questions, the incorrect answers are listed as possible answers. The hint field may be left blank.

At the bottom you may click "Add Another Part" and the fields for another question part will appear. Each question template must have at least one question template part. Clicking the red 'X' next to a question part or field will remove that question part or field. When you are finished, click submit to create your question.

3.3.4 Question Syntax

<eqn> tags are used to specify calculations and random variable assignment within tests. This section details the syntax and commands used in the eqn tags.

3.3.4.1 Variables

Variables are prefixed by the $ character. For example, <eqn $a=3> would give the variable $a the value 3. This will create the variable $a if it wasn't already created, otherwise it will just overwrite the value in $a. Variables can have three types:

- **Number** - May be an integer or a double.
- **String** - Strings must be surrounded by single quotes. The quotes will be removed when the string is printed in the question.
- **NumUnit** - a number together with a unit. Units are made up of a base unit (like m or V) and a prefix (like k or n). For a full list of unit prefixes see section 3.3.4.5. The NumUnit with number 2.1, prefix k and base unit m would be printed as 2.1 km. When a NumUnit is added to or subtracted from another NumUnit, the base units must match or an error will be thrown. There are several functions for manipulating NumUnits detailed in section 3.3.4.3.

3.3.4.2 Binary Operators

Binary operators take two arguments, one on the left and one on the right. For example, <eqn $a+4> would return the value of $a plus 4. + (addition), - (subtraction), * (multiplication), /(division), %(modulo), and =(variable assignment) are supported.

3.3.4.3 EQN Functions

- **format(val, num) –** Returns a string representation of the number val with num decimals. Val can be a Number or a NumUnit. The unit will be included at the end of the string for NumUnits.
- **rnd(n1, n2, (optional) n3) –** Returns a random number between n1 and n2. n3 is the step size (i.e. the random number will be n1 + k*n3 for some integer k). n3 defaults to 1.
- **rndu(n1, n2, (optional) n3, u) –** Returns a random NumUnit between n1 and n2 with unit u. n3 is the step size.
- **rndunit(p1, p2) –** Returns a random unit prefix between the prefixes p1 and p2 (both inclusive).
- **setunit(nu, u) –** Sets the unit of NumUnit nu to unit u.
- **toprefix(nu, p) –** Converts the unit prefix of NumUnit nu to prefix p, and changes the value of nu accordingly.
- **sqrt(n) -** returns the square root of n
- **Trigonometric Functions –** sin, cos, and tan take an angle in radians. The inverse trig functions are prefixed with an 'a'. The degree equivalents of the trig and inverse trig functions end with a 'd'

3.3.4.3 Statements

Statements are expressions to be evaluated made up of constants, variables, binary operators and functions. Operators are evaluated in accordance to B.E.D.M.A.S rules, with functions being evaluated before division and multiplication, and variable assignment occurring last. Parenthesis can change the order of evaluation. Multiple statements, separated by semi-colons, can be executed in one <eqn> tag. The value returned from the last statement will be printed in the question. For example, <eqn $a=cos($b); format($a, 2)> will evaluate the cosine of $b and assign the result to $a. Then it will print $a rounded to two decimal places.

The parser is picky about white space. White space is only permitted (though not necessary) after commas separating function parameters, and semi-colons separating statements. A space is needed between the eqn and the first statement.

3.3.4.4 Other Special Tags

<pic> tags are used to embed uploaded pictures into questions. The syntax is as follows: <pic user/tagname> where user is the username of the user who uploaded the image, and tagname is the image's tag name. For information on uploading files and setting tag names, see section 3.3.8.

The first <_> tag encountered in a question part indicates where the input for entering answers in placed. If no <_> tag is found, the input will be appended to the end of the question part text.

3.3.4.5 Unit Prefixes

The accepted unit prefixes the marker can do unit conversions with are listed here, along with their value when converted to base (no prefix) units.

- 'G' – 10^9
- 'M' – 10^6
- 'k' – 10^3
- ' ' – 1
- 'd' - .1
- 'c' - .01
- 'm' – 10^-3
- 'μ' – 10^-6 (you may type u in place of μ)
- 'n' – 10^-9
- 'p' – 10^-12
- 'f' – 10^-15

3.3.5 Assignments and Practice Tests

There are two types of tests: practice tests and assignments. Practice tests have no assigned or due date, are not worth marks, can be practiced endlessly by students, and do not record any information when students practice them. Assignments have an assigned and due date, are worth marks, and record the student's answers to the questions. Assignments may be practiced like practice tests. There are restrictions on which parts of assignments can be modified once it is assigned, discussed in the next section. A practice test can be assigned, making it an assignment. Assignments cannot be unassigned, only deleted.

3.3.6 Creating a Test

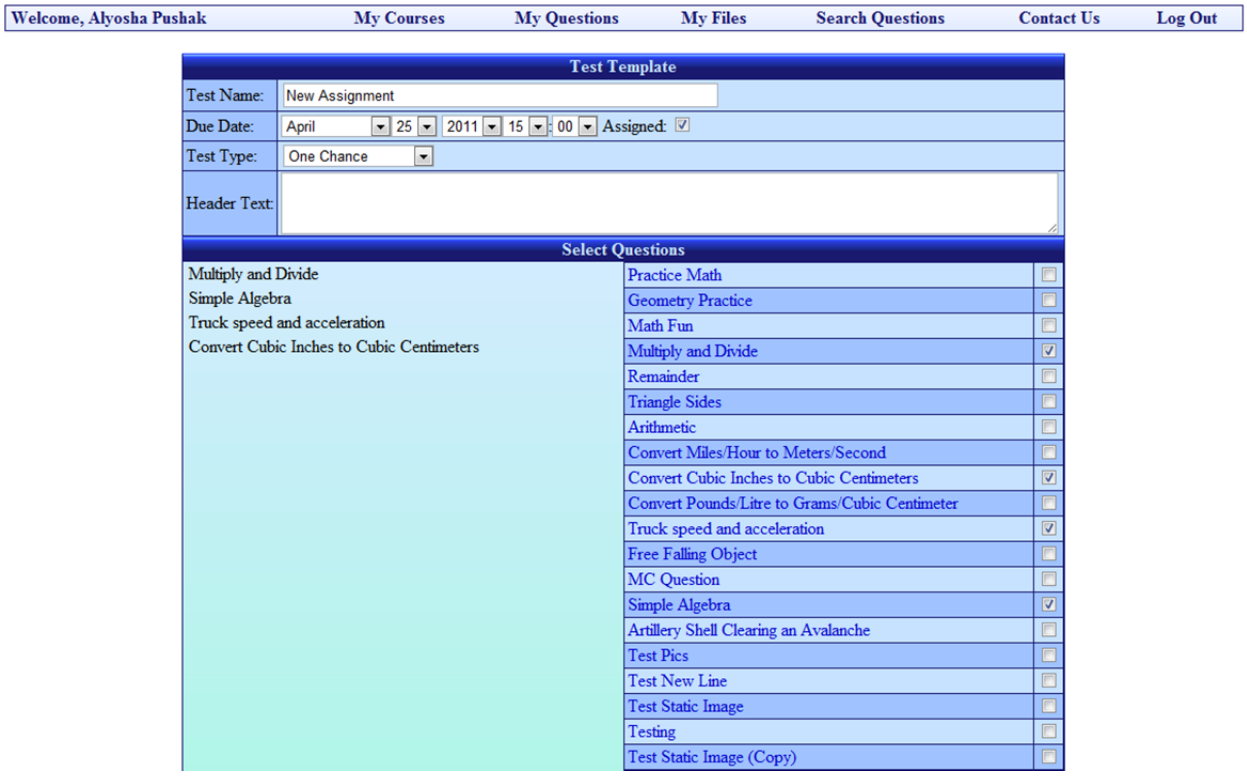

Figure 11: Create a Test

On the course page for the course you want to create a test for, click “Create a Test.” You’ll be taken to the test creation page, where you fill out the following fields:

- **Test Name –** The name of the test. May not be left blank. Max 200 characters.
- **Due Date –** To make the test an assignment, check the assigned check box. You will then be able to give the test a due date. The default due date is one week from the current date, rounded down to the hour. The due date must be at least 24 hours from now.
- **Test Type –** Determines how many times the questions can be answered and how they are answered. The different test types are as follows:
    - **All at Once –** All parts of a question are marked together. Once marked, the answers may not be changed. However, the student may regenerate the question to try again with different values.
    - **One Part at a Time –** Students answer one question part at a time, and must have the previous question part answered correctly before they can answer the next part. They may submit their answers as many times as they like.
    - **One Chance –** All parts of a question are marked at once. Students can only submit their answers once, and may only attempt the test once.

- **Header Text –** Here you can type a message that appears at the top of the test when students practice or attempt the test. This field is optional.

Selecting Test Questions

On the right you'll see the list of questions you've created, with check boxes next to them. On the left is a blank column with a red arrow. The questions that make up the test will be listed here. The red arrow shows where questions will be inserted into the test. Click the check box next to a question on the right, and it will appear on the left where the red arrow indicates. Un-checking a question on the right will remove it from the list on the left. In the left column, the order questions are listed is the order they will appear in on the test. Clicking and dragging a question allows you to change their order. Clicking to the left a question will move the red arrow, changing where questions will be inserted.

3.3.7 Modifying Assignments and Their Questions

To modify a practice test or assignment, on the course assignments page, click "Modify" next to the test name. Once a test is assigned, only the test name, due date, and header text may be changed. When modifying a question that is part of an assignment, a message will appear at the top of the page stating that the question is part of an assignment and if you wish to add or remove question parts, you will have to create a copy of the question. Click on "create a copy" to do so. If you change the marks a question part is worth, all user assignment scores will be updated to reflect the change. A new field called "refresh message" is located at the end of the question template fields. It consists of a check box for the option, "Refresh Active Questions," and a text area, both greyed out. If you edit the question text or any part of a question part besides the marks, the check box will no longer be greyed out. The changes you made will only appear to users who start new attempts of the assignment after the changes are made. Attempts started before the changes will not be changed. However, if you check the "Refresh Active Questions" check box, any assignment questions the user has no marks for will be regenerated to reflect the question changes. Any users who have marks for the modified questions will see a message at the start of the question, saying the question has been changed, and giving them the option to refresh the question to see the changes. If they refresh the question, however, they will lose any answers and marks they had for that question. You may also provide an additional message that will appear below the first, by filling out the text area in the "Refresh Message" field. This allows you to inform students if the change was minor and they don't need to refresh the question, or if it was something they should refresh the question for, such as an answer being incorrect which is now fixed.

3.3.8 Uploading Files

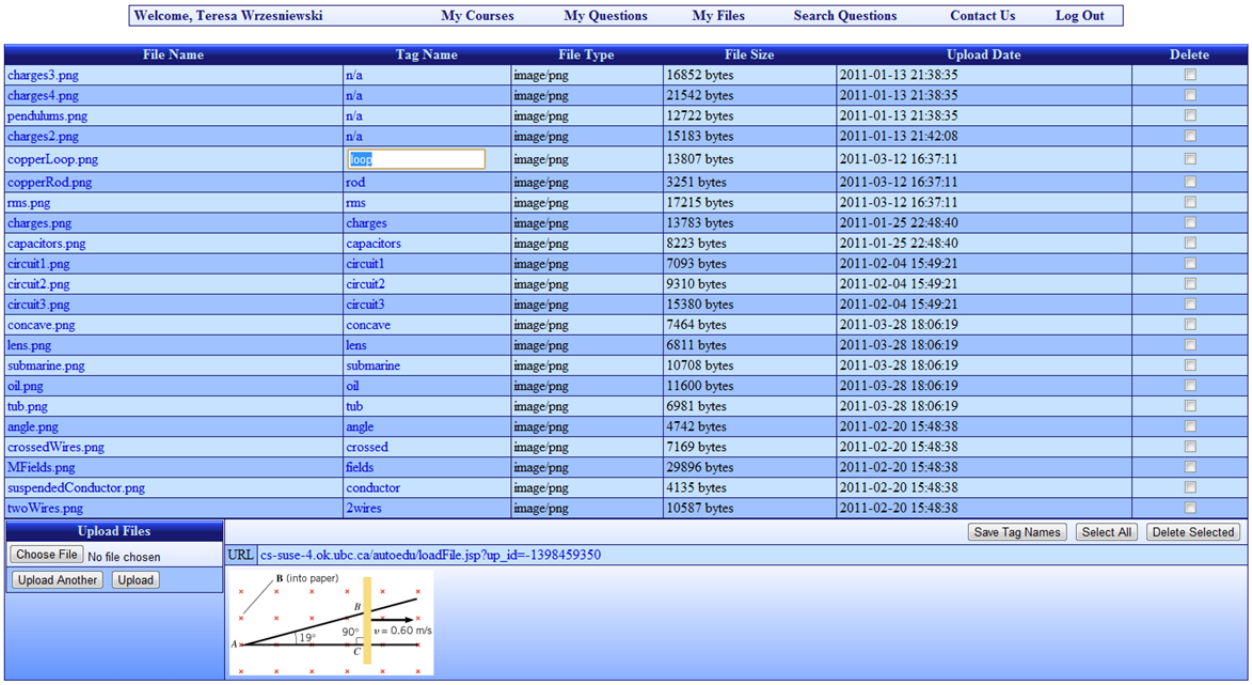

Figure 12: File Upload

From the “My Files” page instructors can upload files. Clicking “Upload Another” allows you to upload multiple files at once. Up to 10 files can be uploaded at once. Once you’ve chosen your files, click “Upload” to begin uploading. When the upload is complete you should see your newly uploaded files added to the list of files. To give a file a tag name, click on the file’s tag name in the tag name column, and type a name in the text box that appears. “n/a” indicates that a file has no tag name. This is the default for new uploads. Once you’ve typed tag names for your files, click “Save Tag Names.” You can use an image’s tag name to embed it in a question. Clicking on a file’s name will display the file’s URL, and, if the file is an image, display the image underneath the URL.

4 Development

4.1 Timeline

August 2010:

- First version of backend done by Dr. Ramon Lawrence. I started working on the project.

September:

- Ability to create and edit question and test templates through the site added.

October:

- First online assignment for PHYS 112.

January 2011:

- First online assignment for PHYS 122.
- Added functionality for marker to do unit conversions between prefixes when marking.
- Question template searching added.

February:

- First online assignment for the second section of PHYS 122.
- Functionality for modifying question templates that are used in assignments added.
- Ability to randomize unit prefixes added.
- Dr. Andis Klergis uses AutoEd for an assignment in BIOC 308.

March:

- Functionality for instructors to override student marks added.

4.2 Challenges

Several challenges and errors occurred during the development process, some of which are not fully understood, for which workarounds were found.

The first concerns loading pictures from the database using jsp. When requesting one picture, the image is consistently loaded properly. However, for a page requesting multiple pictures from the database, some of the pictures often fail to load. To fix this, when our equation solver parses the <pic> tags used to embed images in questions, it assigns a unique id to the picture element that replaces the <pic> tag, and gives the image an onerror event, which reloads the image, identified by its id, when it doesn't load properly.

To use an updated question.jar, tomcat has to be restarted. When tomcat restarts, it erases everyone's session. Anyone who was doing a test at the time would suddenly find all their answers being marked incorrect. To fix this, whenever a question is marked, we check if the session was lost. If it was, the user is informed via an alert box that their session was lost and they'll have to re-login.

During the 3rd online assignment for Phys 112 in term 1, we noticed some students who submitted their test had less than 100%. As the assignment required all questions to be answered correctly before moving on, this should have been impossible. In fact, we observed

test scores changing after the student submitted the test, sometimes days later. The grades were 100% when they submitted, and changed afterwards. Fortunately grades were recoverable – only the overall test score was altered; the marks for each question part were preserved, so the overall test mark could be recovered by summing the individual question part marks. I wrote a java function to do this. I later found a case where a student's answers and marks were overwritten for one question, which was disturbing as the marks can't be recovered in such a case. The student has submitted her assignment so I gave her 100%. I determined that, if a user somehow could view their own test with someone else's test in their session, it would account for all the errors we encountered. If the student hit submit, and the expected answer in their session was different from the test they were viewing, it wouldn't accept their answer. We had some students complaining that their answers weren't accepted. We assumed this was because they weren't formatting their answers properly, or caused by tomcat restarting and wiping everyone's session. If the expected answer in their session matched that for the test they were viewing, when submitting their answer they would overwrite the other student's overall mark, and also the marked date for that question part. I confirmed that the marked dates were after the submission date for some question parts in tests that had their overall mark affected. The last and rarest case, where question answers and marks were overwritten, would occur when the user with the wrong test in their session clicked "Save." This happened very rarely as most students don't use the "Save" button. I was able to view one test with another in my session by opening one test, then opening another test, and then clicking back until I was viewing the first test. I was then able to reproduce all the errors we encountered before. It doesn't make sense for the students to have gotten someone else's test in their session in this way though. We still don't know how the session data becomes inconsistent. To fix the issue, whenever something will be saved to the database, we first check that the correct test is in the session. If not, we reload it. The expected test id is stored in the JavaScript, and sent to the server with all Ajax requests to check against the session's test's id.

Sometimes the formula for correct answers was incorrectly entered into the question template and not caught before the question went into an assignment. The system then rejected student's answers when they were correct. Some of the students this happened to of course found this frustrating. When this happened it was usually caught and fixed within a day or two. Students who tried the question before it was fixed would have to reload the question and redo any question parts they had already answered. Most students were very understanding about this.

Questions sometimes expected the answer to be more precise than the answer the student gave, requiring more significant figures then it should. Precision was changed from 1% to 2% for later assignments to avoid this problem.

5 Results and Feedback

Results were overall very positive. In Dr. Teresa Wrzesniewski's PHYS 112 class during term 1, students were given two written assignments, two online assignments using AutoEd, and a final assignment which they could choose to do on paper or online. The online assignment questions were the same as those for the written assignment, but with some values randomized. In a survey, 70% of students indicated they preferred online assignments to written assignments, and 15% said they liked both equally. However, when given a choice for doing the last assignment online or on paper, 92% of students did it online. So when given the choice, most students who say they liked them equally, and even some who said they preferred paper assignments, chose to do the assignment online. Over 72% of students rated their experiences with the system as satisfactory or very satisfactory. Only 3% rated it very unsatisfactory. 45% indicated they thought AutoEd improved their learning. 81% reported it helped them complete their assignments, and 84% said it helped them hand the assignments in on time. 92% of students thought AutoEd should be used again in Phys122, and 87% thought it could be used in other courses, such as chemistry, math, biology, and even English.

		A1	A2	A3	A4
% Handed-In	2010 – paper	89%	94%	89%	84%
	2011 – online	93%	90%	93%	98%
Average Grade	2010 – paper	60.3%	82.9%	67.3%	62.1%
	2011 – online	84.4%	71.3%	77.5%	86.3%
Median Grade	2010 – paper	67.1%	92.2%	77.9%	75%
	2011 – online	100%	100%	100%	100%

Table 1: Paper Vs Online

Table 1 compares results from the first 4 assignments for the second term PHYS 122 in 2010 when assignments were done on paper, to the results from 2011 when assignments were online. Grades and the percentage of students who completed the assignments improved when assignments were done online for all assignments except the second. The median was 100% each time for the online assignments. This is because students could try answering a question until they got it right, so most students tried for 100%.

A common complaint was the issue of incorrect marking for some answers. This was either because the formula for the answer was encoded incorrectly, or more commonly due to issues with precision and rounding. This was fixed in most cases by increasing the tolerance for accepted answers. For some poorly constructed questions, rounding values part way through calculations could lead to very different results.

6 Conclusions

AutoEd is a valuable teaching tool that can help students learn and dramatically reduce time spent marking assignments, especially in large first year classes. Most of the first year physics students who evaluated the system prefer using it to written assignments. Students especially like the immediate feedback when they submit an answer. They like being able to continue trying to answer a question until they get it right.

7 Future Work

Several ideas and plans were made over the course of the development that, due to time constraints, could not be implemented. I plan to implement these features over the summer.

Currently a copy of a question template must be made for an instructor other than the question template's author to use the template in a test, and test templates cannot be shared across courses. The idea is to allow publishing of question and test templates. Once published, the templates can be used by other instructors. A new table in the database would be created for assignments. Each assignment would point to a test template and course, have an assigned and due date, and an assignment type determining how questions are answered. This way multiple assignments for different courses could use the same test template. Once a template is published, the original author cannot edit it, as other instructors may be using it. A copy must be made instead. To publish a test template, all question templates in the test must also be published.

We want to add the ability to search question templates by keyword. Tags could be added to question templates, which, together with the question text and question part text, would determine the search results.

A mobile version of the site for cell phone browsers is another planned feature.

Dr. Andis Klergis asked how many of his students in BIOC 308 were practicing the online assignment. While we do not currently keep track of this, we could add this feature. Every time a student answers a question in practice or an assignment, we would record the student's id, answer, the question id, and time. Instructors could then view this data to see how many times

each student tries each question, and get an idea of what common mistakes are made. They could then add hints to the questions for those common mistakes.

During summer 2010 I developed a "formula editor" to be used with the AutoEd system. It allows users to type mathematical formula and symbols and have it displayed properly. I used the open source Mathdox Formula Editor developed at Technische Universiteit Eindhoven[5] as a base and tweaked it to suit our purposes. I also wrote a solver for it that can solve equations with functions like sine or log, and even do unit conversions; it can determine if the two terms in 2d * 3a^2/b + 2c (where a, b, c, and d are units) have equivalent units and can therefore be summed. We plan on integrating this with AutoEd, allowing students to show their work and based on that get partial marks for some questions.

Our goal is to continue developing the system and evaluate it in more courses, and eventually release it as open source to the community.

Acknowledgements

I would like to thank Dr. Ramon Lawrence, my honours supervisor, without whom I would not have decided to pursue graduating with honours. The project was his idea. He created the first versions of the database schema and the question.jar, which I modified and used as a base.

I would like to thank Dr. Teresa Wrzesniewski, for evaluating the system in her first year Physics classes. I also owe thanks to Dr. Ashraf Farahat, Dr. Andis Klergis, and Dr. Bulent Uyaniker, who used AutoEd in their classes. Finally, I'm grateful to all the students who tested the system and provided their feedback, and were understanding of the technical errors encountered.

Appendix

Table of Contents

Database Schema

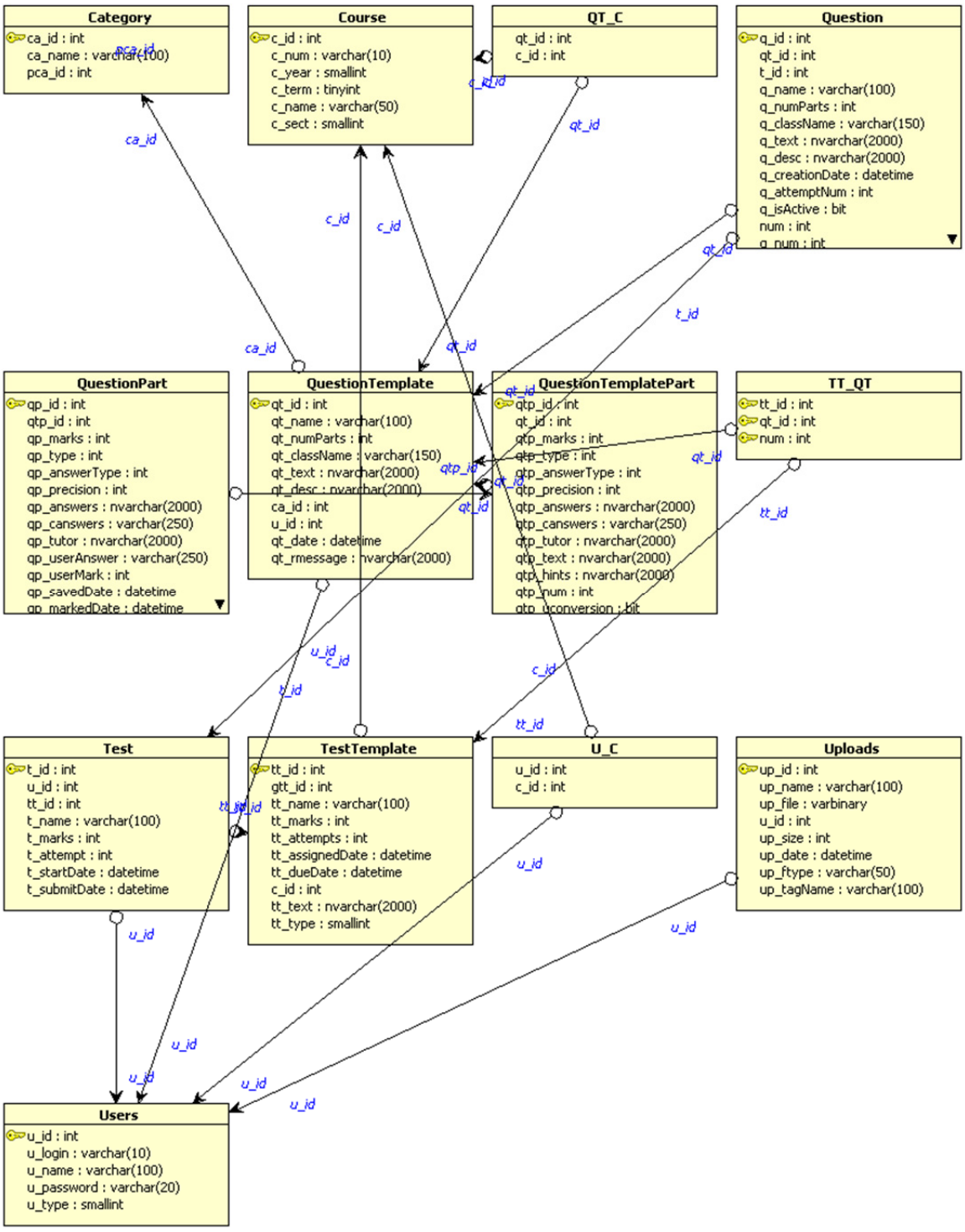

Figure 13: Database Schema

JavaScript Files

HelperFunctions.js

Summary: Provides helper functions for creating form elements and some IE (Internet Explorer) workarounds.

- **getElementsByName(name, tag) –** If the user's browser is IE, this function is added to the document. It returns an array of all elements of the given tag type with the given name. If tag is undefined, it will return elements of any tag type. This function isn't added for browsers other than IE, as they already have this function.
- **setEvent(element, event, f) –** Adds an event to the given element. Event is the trigger, and f is name of the function called by that trigger. Adding events is easy in browsers other than IE, so this is really an IE workaround.
- **createInput(type, name, size, maxlength) –** Creates and returns an input element of the given type, name, size, and max length. Size and max length are optional parameters.
- **createButton(html, onClick, className, name) –** Creates and returns a button element. Html is the inner html; onClick is the function triggered by clicking the button. Class name and class are optional.
- **addOption(select, text, value) –** Adds an option to the given select element with the given value. Text is the text that appears in the drop down menu.
- **selectOption(select, value) –** Selects the option with the given value in the given select element.

QCreator.js

Summary: Provides functionality for the question creating and modifying pages.

- **initQCreator(assignment) –** Call this before using other functions from this script. It just stores a pointer to the element with id "table" for later use, which is the table all the form elements are in. Assignment is true if the question is in an assignment.
- **doneLoad(editing) –** Call this after all question data is loaded via addPart(). Editing is true for the modifying questions page.
- **format(str) –** formats the string stored in the database to the form it's displayed in on the webpage. It converts <quote> to " and <line> to line breaks.
- **adjustButtonPositions() –** It's a hack to keep the red X buttons positioned on the left side of the header bar for each question part, while keeping the text in the center of the bar. It doesn't work in IE so it isn't called for IE, leaving the text on the left instead of the center. It's also not called if the question is in an assignment, as there are no X buttons

in that case as question parts cannot be deleted. It's called every 20 milliseconds to reposition the buttons correctly if the elements are stretched or the page is resized.

- **change() –** Called when the user changes a value in a question part other than marks. This will cause the existing questions parts to be written over in the database. The changes will only show in questions generated after the changes are saved.
- **changeMarks() –** Called when the user changes the marks for one of the question parts. This will trigger a recalculation of all the user's marks for questions generated from this template once the changes are saved, which could change their marks on tests that contains those questions.
- **addAnswer(num, answer) –** Adds another input element for entering another correct answer to question part #num. If answer is defined, that answer will be put in the input element, as is the case when loading the initial question data on the modify question page.
- **removeAnswer(qnum, anum) –** Removes the input element for a correct answer. Anum is the answer number, qnum is the question part number. If that question part has only one correct answer, it does nothing, as question parts must have at least one correct answer.
- **addHint(num, answer, hint) –** Adds form elements for adding another incorrect answer with a hint to a question part. Num is the question part number, answer and hint are the initial values entered in the form elements.
- **removeHint(qnum, anum) –** Removes form inputs for incorrect answer and hint #anum from question part #qnum.
- **addPart(qpmarks, qptype, qpprec, qptext, answers, wrongAns, hints, uconv) –** Adds another tbody containing form elements for entering another question part. The parameters are the initial values for the question part.
- **removePart(num) –** Removes the tbody containing all the form elements for question part #num. If there's only one question part, it does nothing, as all questions must have at least one question part.
- **addPrecision(num) –** Called when the answer type of a question part is changed. It checks if an input element for precision or a checkbox for allowing unit conversions needs to be added or removed.
- **addRow(text) –** Adds another row to the tbody of the question part currently being added. Text is the text that appears in the left title column. Returns a pointer to the td of the right column.
- **validateForm() –** Validates form input before submitting the form and creating or modifying a question. To be valid, the following is required:
- The question must have a name.

- The question text, question description, and each question part text must be 2000 characters or less.
- Marks for each question must be an integer greater than 0.
- Question parts with precisions must have an integer precision between 0 and 100.
- All Question parts must have at least one correct answer.

Dialog.js

Summary: Create custom dialog boxes. Like the pop-ups that appear with alert messages, but customizable.

- **Dialog(width, text) –** Returns a Dialog object. Width is the width of the dialog box, and text is the header text. This prepares the html elements for the dialog box, but does not display the dialog box on the page.
- **Dialog.createTD() –** Creates another row (TR and TD element) in the dialog box and returns the TD element for that row. Rows are separated by lines. Custom content can be displayed in the TDs.
- **Dialog.close() –** Removes the dialog box from the page. The html elements are preserved in the Dialog object so the dialog box can be reopened.
- **Dialog.show() –** Shows the dialog box. Call this after all custom content is added to the dialog box. The box is centered based on its current size. It will not recenter if it's sized is changed after it's opened. While a box is open, clicking off the box will cause the box's header to flash, and clicking links will not direct you to another page, and form elements will not focus. The variable dbox is true while the dialog box is open. This can be used to disable other clickable events. Header flashing is disabled for the first 100 milliseconds after opening the box to prevent flashing from the initial click that opened the dialog box.
- **Dialog.flash() –** causes the dialog box's header to flash. If called with Dialog.count = 0, it will flash 6 times, once every 150 milliseconds.
- **enableFlash() –** Enables dialog box header flashing. Called 100 milliseconds after a box is opened.
- **handleClick(e) –** Causes the dialog box's header to flash by calling Dialog.flash() with Dialog.count = 0. Called when the user clicks off the dialog box.

Categories.js

Summary: Creates a dialog box for selecting a category. Categories have parent categories. The root categories are listed. Clicking the + sign next to a category shows all the children categories of that category. Clicking the – sign then hides the children categories. Uses dialog.js.

Variables:

- **categories –** An array. categories[id] contains a list of all children categories of the category with that ca_id.
- **cats –** An array of categories indexed by ca_id.
- **dia –** The dialog box.
- **display –** The div in the dialog box displaying the categories.
- **divs –** An array of divs containing a group of sibling categories.

Functions:

- **Category(name, id, pid) –** Returns a Category object. Name is the category name, id is the ca_id, and pid is the pca_id (parent id). Pid is 'n' for no parent.
- **addCategory(name, id, pid) –** Creates a Category object with name, id, and pid, and adds it to the cats array, and the appropriate location in the categories array.
- **displayCategories() –** Opens the dialog box displaying the categories. Will create the dialog box dia if it was not already created. When initially opened, all root categories will be displayed, with no categories expanded.
- **insertCategory(id, div) –** Adds the category with the given name to the given div. Div should be a div containing a group of sibling categories. The + character for showing the children categories of the category will be added to the end of the div if the category has any children, along with the category name.
- **expand(id) –** Expands or closes the category with the given id, showing or hiding all its children categories (depending on if it's currently expanded). If this is the first time expanding the category, a div will be created that holds all the children categories and added below the parent category. The div has 20 pixels of padding on the left to indent the list of children. For subsequent expansions the div is reinserted below the parent. The div will be restored to the same state it was in when removed, so children categories that were expanded will remain expanded. When closing a category the div with its children is removed. Called when a + or – character is clicked.
- **chooseCategory(id) –** Selects the category with the given id. Closes the dialog box and fills the textbox with id "category" with the selected category name. Called when a category name is clicked.

MiniSolver.js

Summary: A simple calculator than can solve equations made of binary operations (+, -, /, *, ^), numbers, and parenthesis. It is a simplified version of a solver I wrote in summer 2010 for another project.

The Algorithm: We begin parsing the equation string from the left. We have a stack called parse, onto which we push numbers and operators. If the character we are parsing is a number or a '.', we append to a string called number. If not, we call pushNumber(). This converts the string in number to a number and pushes it on the parsed stack, then sets the string number to an empty string. If the string number could not be converted to a number, it throws an error. Any parsing errors terminate the parsing and send an alert message to the user describing the error. Before pushing the number on the stack, we call checkInvisibleTimes(). If the previous object on the stack was a number, this will push a * (multiplication) operator on the stack. This can happen in cases such as: "4(5)2". If our parsed character is a '-', and the last object on the stack was an operator or a '(', we append the '-' to the number string, making the next number a negative number. When we parse an operator, we may be able to evaluate part of the expression. If the operator we parse has precedence lower than the last operator we parsed, we start popping objects off the stack and evaluate them until we reach an operator with precedence less than that of the operator we just parsed. Then we push the resulting number from our evaluation onto the stack, followed by the operator we just parsed. For example, if we parsed 3+-5*6^2*, once we see the second '*', we know we can evaluate the -5*6^2. After evaluation our stack will have 3+-180*. If we parse a '(', we first check our number string. If it is equal to '-', we push a -1 on the stack and empty the number string. Next we call checkInvisibleTimes(). The we evaluate off the top of the stack until we encounter a '('. White space is allowed between numbers and operators. If we parse a character we don't know what to do with, that character is assumed to be the start of the units, parsing is stopped and we evaluate everything on the stack.

Question.js

Summary: An abstract class representing a question part. There are different question classes for the different types of question parts (text, multiple choice), which implement this class.

Variables:

- **id –** The id of this question. The input element(s) have this id, or name if there are multiple elements (for multiple choice).
- **mark –** The user's mark for this question part.
- **score –** Total marks this question part is worth.
- **solver –** a boolean indicating if the question part answer type is text with solver. Only the text question part class has this field.

Function:

- **getAnswer() –** Returns the user's answer. If this is a text type with a solver and the answer begins with an '=', the answer string will be passed to miniSolver.js to solve the equation.
- **clearAnswer() –** Clears the user's answer.
- **lock() –** Locks the question part, disabling and greying-out the input for answering. If the user hasn't answered this question yet and the entire test isn't locked, it gives the input an onfocus event that displays an alert message informing the user they must answer the previous question part correctly before answering this one.
- **unlock() –** Unlocks the question part, so it can be answered.

Test.js

Summary: Contains methods shared by the pages for doing assignments, practice tests, and practice questions.

Variables:

- **type –** The type of the test, which effects how questions are answered. See section 3.3.6.
- **questions –** An array containing the questions.
- **qmarks –** An array containing the user's mark for each question.
- **score –** The user's total mark for the test.
- **numLocked –** The number of locked questions.
- **submitted –** true if the test was submitted
- **assignment –** true if the test is an assignment.

Functions:

- **checkAnswer(qpid, answer) –** Checks the answer for the question part with the id qpid. The marking is done on the server by sending an AJAX request to markQuestion.jsp. Nothing is recorded in the database. If the test is an assignment, it checks the due date before marking the question, and gives an alert message if the due date is passed. If the session was lost (either due to session timeout or the server was restarted), the server responds saying the session was lost, and an alert message informs the user that the session was lost and they will have to log in again. A checkmark is displayed next to the answer if it is correct, whereas a red ex appears if it is not. This also displays the "Show Hint" button if a hint is available. It ends by calling addmarks for the appropriate question, so the question's marks and the marks in the test header are updated.

- **format(str) –** returns the string str with all single quotes replaced with the right-single quote special character.
- **reloadImage(id) –** reloads the image with the given id. Called when an image fails to load.
- **showHint(hint, id) –** Shows the hint text hint in the element with the given id. Called when the "Show Hint" button is pressed.
- **incLocks(qnum) –** locks the question with number qnum, increases the total lock count, and enables assignment submission if all questions are locked. Locked questions can't have their answers edited.
- **markQuestion(qnum) –** Marks the question with number qnum. Behaviour differs depending on the test type and whether or not the test is an assignment. All answers are checked through calls to checkAnswer. For assignments, answers, marks, and whether or not the question is active is saved though AJAX requests to saveQuestion.jsp. For some test types incorrect answers do not save anything to the database.
- **lockQuestion(qnum) –** Locks the question with the given qnum. It is different from incLocks in that it locks each answer input and displays a green check mark or red ex for each question part depending on if the user has marks for that question part, whereas incLocks assumes these things have already happened. It finished by calling incLocks(qnum). This is called to lock an inactive question when the page is initially loaded. incLocks is called when a question is first locked because the user just submitted their answer to it.
- **partialLockQuestion(qnum) –** Used for test types that require the question parts to be answered one at a time, in order. Locks all the question parts that the user currently cannot answer or has already answered. Will lock the entire question and call incLocks(qnum) if all the question parts are answered.
- **lockAll() –** Locks all questions. Called when viewing a submitted test, a test passed the due date, or when an instructor is viewing a student's test.
- **clearQuestion(qnum) –** Depending on the test type, either clears the answers to all question parts for the question with number qnum, or just the active question part. Called when "Clear" is clicked.
- **saveQuestion(qnum) –** Saves user answers to all parts of the question with number qnum, though Ajax requests to saveQuestion.jsp. The question header will say "Save Successful" when the save is completed successfully or "Save Unsuccessful if an error occurred.
- **regenQuestion(qnum) –** Regenerates a new instance of the question with number qnum through an Ajax request to regenQuestion.jsp. Called when the retry button, which appears for some test types, is clicked, or when the user clicks refresh for a question the instructor modified since the question was initially loaded. The user's

answers are cleared, and the user's marks for the question are reset to zero. If the question was previously completely answered, the total number of locks is decremented, which may disallow test submission if all questions used to be answered.

- **override(qnum, qpnum) –** Hides the "Override Mark" button and shows the form inputs for an instructor to override the student's mark for question part qnum of question qnum. Called when an instructor clicked an "Override Mark" button.
- **setOverrideMark(qnum, qpnum) –** Sets the initial value of the input for overriding a mark to be the user's current mark for that question part. For some reason changing the value when the input is first appended to the document in override() doesn't work. Calling this function a millisecond after override() does.
- **doOverride(qnum, qpnum) –** Saves the overrided mark and answer to question part qpnum of question qnum, and the new overall test mark through an Ajax request to saveQuestion.jsp. Hides the form inputs for changing the mark and brings back the "Override Mark" button, and changes the user marks in the question header and the test header.
- **addMarks(qnum) –** Sums the user's marks for each question part of question qnum and updates the marks in the question and test headers accordingly.
- **scrollToNum(num) –** Scrolls the question with number num. Called when a question number is clicked in the test header.
- **calcTime(time, id) –** displays the time in the form "mm:ss" in the element with the given id. Time is the time to be displayed in seconds.
- **keepTime() –** Keeps track of the total time the user spends on the page, and updates the time shown in the test header every second through calls to calcTime. Timing stops when all questions are locked. The time starts at 0 each time the page is loaded.

Flot API

Flot is an open source JavaScript plotting library [1]. It was used for creating the graphs of assignment data on the "View Results" page.

jQuery

"jQuery is a fast and concise JavaScript Library that simplifies HTML document traversion, event handling, animation, and Ajax interactions" [2]. The Flot API requires jQuery. Asides from the page using Flot, jQuery was not used.

Java Files

Header.jsp

Summary: Contains methods for printing the header that appears at the top of each page, and the course header appearing below the header on all course pages. There are two versions of each method, one with a String path parameter for passing the path to the directory containing header.jsp. The other with no path arguments assumes we already in the correct directory. Printing the course header requires the course id as an argument.

MarkQuestion.jsp

Summary: Send an Ajax request to this page to mark an answer.

Parameters:

- **mytext –** The answer to be graded.
- **num –** The question number followed by the question part number, separated by an underscore.
- **tid –** The expected test id. If the test in the session does not have this id, the test is reloaded. Can be null (when marking for a practice test or question that is not saved in the database).

SaveQuestion.jsp

Summary: Send an Ajax request to this page to save data to the database. The data saved depends on the parameters.

Parameters:

- **tid –** The expected test id. If the test in the session does not have this id, the test is reloaded.
- **num –** The number of the question to save. Optional.
- **pnum –** The number of the question part to save. Optional. If num is provided and pnum is not, all parts of the question are saved. If pnum is given, ans must be as well.
- **ans –** The answer to save over the old answer for question part pnum. Optional, but must be provided if pnum is.
- **mark –** The overall test mark to save over the current test mark. Optional.
- **deactivate –** Optional. If set to anything, the question num will be deactivated.

RegenQuestion.jsp

Summary: Send an Ajax request to this page to regenerate (reload with new random values) a question.

Parameters:

- **tid –** The expected test id. If the test in the session doesn't have this id, the test is reloaded.
- **num –** The number of the question to be regenerated.
- **marks –** The total number of marks the question is worth (sum of the marks for each question part).

UploadBean

UploadBean is a Java component for uploading files from browsers [3]. It handles our file upload requests to the database.

JavaCC

JavaCC is a parser generator for Java[4]. It was used to generate the parser for evaluating <eqn> tag expressions.

Question.jar

All class representing an object that gets saved to the database (templates, questions, tests, etc.) have a save(Connection con) and load(int id, Connection con) method. Save saves the object to the database, along with any objects that belong to that object (for instance saving a question saves all its question parts). If it's not already inserted into the database, it will insert it. Save() returns the id of the object. Load will load the object with the given id from the database. Each object stores variables corresponding to the attributes stored in the database. Their Ids are -1 If they aren't saved to the database.

TestTemplate

Summary: Represents a test template.

Methods:

- **ArrayList<Test> loadUserTests(int userId, Connection con) –** Loads all the test instances of this test template started by the user with the given id, and returns them in an ArrayList.

- **Test generate() –** generates and returns a test using this test template. To generate a test, all questions that make up a test must be generated from their respective question templates.

This class also contains various getter and setter methods, and a toString method.

Test

Summary: Represents a test. Extends TestTemplate.

Methods:

- **void addQuestions(Question q) –** Adds question q to the ArrayList of questions for this test.
- **void submit(Connection con) –** sets the submit date to the current time and saves the test.
- **saveMark(Connection con, int mark) –** sets the user's mark to the given mark, and saves that mark to the database.

This class also contains various getter and setter methods, and a toString method.

QuestionTemplate

Summary: Represents a question template.

Methods:

- **void addPart(QuestionTemplatePart qtp) –** adds qtp to the ArrayList of question template parts.
- **void insertPart(QuestionTemplatePart qtp, PreparedStatement pstmt, int qtid) –** Saves qtp to the database with question template id set to qtid. Called by save(Connection con).
- **Question generate() –** generates a question from this template. Question generation involves generating all of the question's question parts from their respective question template parts.

This class also contains various getter and setter methods, and a toString method.

Question

Summary: Represents a question. Extends QuestionTemplate.

Methods:

- **Question(QuestionTemplate qt) –** Constructor. Generates this question from the template qt by calling generateFromQT(qt). Question text is evaluated here.
- **generateFromQT(QuestionTemplate qt) –** Generates this question from template qt. Question text for each question part is evaluated here. Do not use this method by itself to generate a question. Use the above constructor.
- **void saveAnswers(Connection con) –** saves answers to all of the question parts, via calls to saveAnswer().
- **void saveAnswer(int partNum, Connection con) –** saves the answer to question part number partNum. Part numbers start at 1.
- **String mark(int partNum, String answer) –** Marks the answer for question part partNum. Saves nothing to the database. The result of the marking is returned as a String, which will include any hints if there are any.
- **int regenerate(Connection con) –** Regenerates this question (reloads with new random values). Save the question to the database and returns the question id.
- **saveActive(Connection con, boolean isActive) –** Saves whether or not this question is active.

This class also contains various getter and setter methods, and a toString method.

QuestionTemplatePart

Summary: Represents a question template part.

Methods:

This class contains various getter and setter methods, and a toString method.

QuestionPart

Summary: Represents a question part. Extends QuestionTemplatePart.

Methods:

- **QuestionPart(QuestionTemplatePart qtp) –** Constructor. Generates this question part from the template qtp.

- **Regenerate(QuestionTemplatePart qtp) –** Regenerates this part using template qtp (reloads with new random values).
- **String mark(String answer) –** Marks the given answer with a call to markAnswer(answer). Sets the markedDate and the savedDate to the current date. The results of marking are returned as a String.
- **int getUnitsLoc(String s) –** Returns the start of the units in s. Ignores the first E followed by a '-' or a digit, as that indicates scientific notation and is not a unit. Returns the length of s if s contains no units.
- **String extractUnits(String s, int unitLoc) –** Extracts the units from s starting at unitLoc and returns them as a String.
- **int extractInt(String s, int unitLoc) –** Extracts and returns an int from s, ending at unitLoc.
- **double extractDouble(String s, int unitLoc) –** Extracts and returns a double from s, ending at unitLoc. Converts from scientific notation is s is of the form 2.1E-3.
- **String markAnswer(String answer) –** Marks the given answer but saves nothing to the database. The appropriate extract methods are used to extract the answer (int, units, etc.) from the answer String, and the appropriate method in the marker class is called, according to the answer type. The answer is checked against each answer in the ArrayList of answers. The results of marking are returned in a String, including any hints that go with that answer.

This class also contains various getter and setter methods, and a toString method.

References

[1] Laurson, Ole (2011). *Flot – Google Code.* http://code.google.com/p/flot/

[2] jQuery (2010). http://jquery.com/

[3] Java Zoom. *UploadBean Support.* http://www.javazoom.net/jzservlets/uploadbean/uploadbean.html

[4] Oracle (2011). *JavaCC.* http://java.net/projects/javacc

[5] Technische Universiteit Eindhoven(2007-2008). *MathDox Formula Editor.* http://www.mathdox.org/formulaeditor/

[6] WebAssign. http://www.webassign.net

[7] Aplia. http://www.aplia.com

[8] Mastering Physics. http://www.masteringphysics.com

[9] Gradiance. http://www.gradiance.com

[10] Carnegie Learning. http://www.carnegielearning.com

[11] Blackboard. http://www.blackboard.com

[12] Quest. http://www.quest.cns.utexas.edu

[13] LON-CAPA. http://www.lon-capa.org