Extending and Contributing to an Open Source Web-based System for the Assessment of Programming Problems

Olly Gotel
Pace University
Seidenberg School of
Computer Science and
Information Systems
Computer Science
New York, NY 10038
ogotel@pace.edu

Christelle Scharff
Pace University
Seidenberg School of
Computer Science and
Information Systems
Computer Science
New York, NY 10038
cscharff@pace.edu

Andy Wildenberg
Cornell College
Dept. of Computer Science
Mount Vernon, IA 52314
awildenberg@cornellcollege.edu

ABSTRACT

This paper describes the development of a web-based programming and assessment environment for use in supporting programming fundamentals courses (CS1, CS2) taught in Java. This environment is currently linked with WeBWorK (a course management system (CMS) that is popular for administering and assessing mathematics and physics coursework) but is designed for the potential integration with other CMS environments. In addition to the traditional multiple-choice and short answer questions that have been augmented with the extensive randomization and customization routines of WeBWoRK, this new environment (called WeBWorK-JAG), can automatically collect and grade free-form program fragments written in Java. Novel pedagogy has been developed based on the capabilities of this extension and preliminary classroom results are discussed in this paper. For example, when students learn to create WeBWorK-ready problems for their peers, they are exposed to the reality of creating comprehensive unit tests and to the wider quality assurance aspects of formulating problems and their solution sets. This work is described in the context of an emerging commercial market for online programming assistants and the unique contributions of this new environment are summarized.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer Science Education

General Terms

Automated Grading, Eclipse, Java, JUnit, Open Source, Peer-review, Testing, WeBWoRK

1. INTRODUCTION

There is a growing proliferation of web-based systems to support the practice and assessment of programming tasks for educational purposes [5]. This paper describes work carried out to extend WeBWorK [11, 13], an open-source course management system (CMS) that has strong facilities for understanding mathematical formulæand randomization capabilities, to be useful in the field of Computer Science for this purpose. Being open-source and hence customizable, we propose that using WeBWorK as a platform for such a programming assessment system offers many advantages over existing systems. Our preliminary efforts and experiences in adapting WeBWorK to Computer Science are described in previous papers [1, 2, 15]. The main contribution of this new paper is a description of an extension to WeBWorK that was created to allow the system to automatically grade Java program fragments in real time, thereby permitting more flexibility in the type of questions that WeBWorK can support. This extension is referred to as WeBWorK-JAG (Java Auto Grader). The paper also describes our experiences with the use of this extension in the classroom, particularly to support alternative approaches to pedagogy, such as having students contributing their own questions to the system library.

The paper is organized as follows: Firstly, we summarize the educational context for this work in Section 2 and then we provide a brief critique of existing systems to support the online assessment of programming problems in Section 3. A synopsis of WeBWorK is provided in Section 4 and the salient distinctions over comparable systems are highlighted. The JAG extension to WeBWorK is described in Section 5. Section 6 explains the process of writing WeBWorK-ready programming questions for assessment purposes and motivates the educational potential of student problem formulation. In Section 7, we describe the use of WeBWorK in the classroom and compare student experiences with using both vanilla WeBWorK and WeBWorK with the JAG extension. By getting students to create, design and implement their own robust WeBWorK questions, we found important and useful lessons for pedagogy. These are highlighted in Section 8 and address requirements for teaching programming that we set out in Section 2. We also found this approach an expedient way to expand our library of WeBWorK questions. We end the paper with a summary of our future plans in this area.

2. PEDAGOGICAL CONTEXT

Learning to program is fundamental to Computer Science education. It is one of the first skills that an undergraduate student is expected to master [16, 17]. However, it is a skill that can take time and practice to acquire, and many students need guidance and feedback beyond that which can be provided in the traditional classroom. In response, a number of web-based systems for the delivery and automatic assessment of programming assignments have been developed, a representative set of which is described in Section 3. Such systems aspire to augment educational practices rather than to supplant them. Prior to describing the features that enable this in such systems, this current section outlines some of the broader classroom practices that we were seeking to both support and enhance that influenced our selection of a programming assessment system for use in our CS1 and CS2 programming fundamentals courses.

Open Source and Industrial Relevance. Not only is the use of open source software appealing to small schools, its use begins to teach students about the speed and nature of change in the software industry. Additionally, early exposure to the results of collaborative and community-driven development illustrates the importance of using shared coding standards and possibly provides a ready context for later contribution to the open source movement. When using such software to support core courses, students are taught to first check the 'frequently asked questions' (FAQs) and mailing lists if they are experiencing any problems with the software per se. Our non-empirically validated experiences to date indicate that this can avoid some of the dependence on the instructors as a first port of call for problems and help cultivate independent problem-solving behaviors, behaviors that likely directly translate to programming competence. For this reason, Eclipse *(http://www.eclipse.org)* is used in our classrooms as the integrated development environment (IDE) of choice. Students need to use a programming assessment system that aligns with the philosophy and practical principles that courses are attempting to cultivate.

Requirements and Testing. Although profiled as a core practice of the agile community [3], test-driven development (or test-first) has been a long-standing practice of the programming community. By careful attention to how a program or program fragment is to be evaluated for correctness, students learn about writing quality code that passes stringent tests. In the process of writing the tests, they learn more about the problem and its solution, so about the criticality of formulating requirements in a testable manner, thereby laying the basis for quality work. Students also learn about important mechanisms that can help them to assess and manage the impact of subsequent changes and so the concept of regression testing. Test-first is integrated into our courses from an early stage and supported by JUnit. However, we have faced problems in getting students to formulate thorough tests and to appreciate the value of so doing. In addition to learning to write the code itself, students need to use a programming assessment system that fosters a greater appreciation for requirements and testing activities.

Peer review. Code inspections or structured walkthroughs are another industrially practiced discipline that that we strive to implement in the classroom [10]. This gives students an awareness of the value of getting an independent person to examine code and detect errors prior to release and use by others. It is a central practice of software quality assurance [14]. Any programming assessment system that a student uses needs to provide the means to emphasize and practice these complementary skills, else the message surrounding the value of doing this in practice, something that is hard taught in the classroom, is simply not transferred into personal practice.

Programming as more than Coding. There are many advantages in giving students online practice questions in a programming fundamentals class and in their automatic electronic grading. For students, assuming the questions are aligned with the course material and well written, there is the benefit of instant responses and possible feedback. This can encourage safe experimentation in a more timely fashion than the typical student-instructor homework review cycle. For the instructor, the time spent in marking can be better spent in crafting questions and in picking up on critical issues in individual student responses or across the class more generally, thus re-focusing teaching efforts. Despite all the benefits, we contend that many existing systems for the automated assessment of programming problems focus purely on the coding aspect. To augment our courses, we require a system that can be used to integrate the above mentioned practices into its everyday use.

3. EXISTING ASSESSMENT SYSTEMS

In the last five years, a number of systems for the delivery and automated assessment of programming assignments have been developed. Questions supported range from true / false, multiple-choice and matching questions to program writing. Assessing a computer program involves measuring its correctness, quality and authenticity. In this context, correctness refers to the extent to which a program's behavior matches its specification and is achieved by testing, quality covers code typography, and authenticity includes student identity and plagiarism prevention. Existing systems focus mostly on correctness and include increasingly advanced testing approaches based on shell scripts or JUnit. A selection of some of the early systems can be found in a journal issue dedicated to the topic [5]. While there is a great variety in the field, they can generally be grouped by whether they are commercial (with or without the possibility to author new questions), or free and open-source systems.

Commercial Systems. Commercial and closed systems come with a fixed set of questions, generally associated with a textbook and publisher, meaning that the systems are neither free (i.e. having a per-user cost) nor general purpose. Other systems are extensible in the sense that instructors can create homeworks by picking and choosing questions in existing libraries. The teaching pedagogy for Java being rather individualized, it is difficult to write libraries that would be as widely useful as, for example, in calculus or introductory physics, where the different pedagogies are much

more thoroughly established.

Proprietary web-based systems, such as CourseMarker (*http://www.cs.nott.ac.uk/CourseMarker*), CodeLab (*http://www.turingscraft.com*), MyCodeMate (*http://www.mycodemate.com*), OWL (*http://owl.course.com*) and Viope (*http://www.viope.com*), focus on the delivery of Java, C and C++ problems. MyCodeMate and OWL are very popular in the US and are associated with specific textbooks. CourseMarker is based on formative assessment; it allows intermediary submissions with feedback. It has a powerful marking engine that grades programming assignments with respect to correctness and lexical / typographic analysis. CodeLab delivers only short answer exercises, focusing on a particular programming construct or idea. It is defined by its authors as a "context for practice". Using some very sophisticated data mining techniques, it provides student users with error highlighting of their code and hints on how to correct it. MyCodeMate supports the compilation and execution of code but does not check for correctness, being seen more as a repository. CodeLab and OWL offer graphical-based progress and performance visualization. Viope is the only system offering a programming editor with syntax highlighting to submit code. Gradiance (*http://www.gradiance.com*) is a proprietary formative assessment system that delivers database, compiler, automata and language theory, and operating systems problems. In Gradiance and MyCodeMate, if a wrong answer is submitted, an explanation as to why is provided.

Free / Open-source Systems. There are very few open-source or free assessment systems. QuizPACK (*http://www2.sis.pitt.edu/ taler/QuizPACK. html*) is a free system that delivers parameterized matching questions focusing on tracing C programs / fragments. BOSS (*http://www.dcs.warwick.ac.uk/boss*) is a free summative assessment system that can measure correctness, quality according to a target coding style and authenticity. Very recently, JavaBat (*http://javabat.com*) was released. JavaBat permits students to practice Java coding online. Instructors can create classes and view the summary tables of their students' progress.

The study of the existing systems will inform our future development of WeBWorK. As open-source software, WeBWorK is fully extensible, and its powerful authoring language (allowing it to have complete parameterization and interface with other software) could address a wider range of problems associated with programming than some of the other systems.

4. WEBWORK

WeBWorK (*http://webwork.math.rochester.edu*) is an open source web-based assessment tool and course management system (CMS) that can create, deliver and automatically grade homework problems in a variety of subjects, including calculus [11, 13, 19], physics and discrete mathematics [18]. Originally developed at the University of Rochester, it is in wide use in academia; current estimates show it in use at about 120 universities and colleges world wide, as well as several high schools. WeBWorK's goals are "to increase the effectiveness of traditional homework as a learning tool by providing students with immediate feedback on the validity of their answers, giving students the opportunity to correct mistakes while they are still thinking about the problem," [12] and to encourage active learning, capitalizing on promising research on the relationship between on-line feedback and academic performance [4, 6, 7, 8], as well as on the role of re-testing opportunities in mastering learning [9].

WeBWorK's strength and popularity stems from its use of an advanced problem generation language called *Problem Generating* (PG), a macro language that mixes Perl, LaTeX, HTML, and text. PG permits the drawing of graphs and functions, and it can also be extended to use JavaScript and interface with Java Applets, making WeBWorK an extensible platform. In WeBWork, all questions (and answers) are written in PG. Boilerplate examples allow authors to write standard multiple-choice, matching, true / false, short answer and numerical comparison questions. Because each question is in essence a (sometimes very simple) program, question authors can write questions that use randomization so the particular instance of a question that each student sees is unique. For example, a question asking students to identify the median of a list of 9 numbers could actually generate a completely different list of numbers for each student in the class. Similarly, an instructor could write the question "What is the derivative of $ax^2 + bx + c$", specify the answer was $ax + b$, ask WeBWorK to choose suitable values for a, b, and c, and the resulting question would now be different for every student in a class.

WeBWorK also includes a large set of *answer evaluators* that determine when an answer is correct or not. Commonly used evaluators are string matching (with or without regular expressions) or numerical matching. More advanced evaluators can determine whether two trigonometric / polynomial functions are equivalent, so that if the answer to a question were 1, then $\sin^2 x + \cos^2 x$ and $\frac{(x+1)(x-1)}{x^2} + x^{-2}$ would also be recognized as correct answers. Questions may be typeset using a generous subset of LaTeX, and students may enter their answers using LaTeX as well. This ability to typeset and understand common mathematical functions has made WeBWorK extremely popular in the mathematics and physics community.

5. WEBWORK-JAG

Vanilla WeBWorK, while it provides support for formulating a variety of types of questions such as true / false, multiple-choice and short answer type questions, does not provide support for free-form programming questions, as per some of the systems described in Section 3. WeBWorK-JAG was explicitly developed to address this limitation and to hence extend the capabilities of WeBWorK for use in Computer Science. Our WeBWorK server is accessible for full experimentation at *http://atlantis.seidenberg.pace.edu/webwork2.*

The architecture of WeBWorK-JAG is designed to integrate into WeBWorK in a simple manner, yet be sufficiently general and modular to be hooked into other CMSes. In WeBWorK questions, authors specify three things in the PG files: the text of the question (which often includes variables for randomization purposes), the correct answer for the question (which of course is a function of any randomization variables), and which *answer evaluator* is the correct one to

use to determine whether the student's answer is equivalent to the correct answer.

The JAG extension changes this structure slightly. The PG files still contain the statement of the problem (though this is really best considered a problem specification). The answer to the problem is encoded in the JUnit class associated with the template class, so it does not need to be specified in the PG file. A new "java answer evaluator", called when a student submits code, was created that performs all the necessary tasks to determine how correct a submission is.

To evaluate the correctness of the submission, the following is executed:

1. A new temporary directory is created, and all the files present in the "code" directory for the problem are copied into it.
2. The template file is expanded by replacing the marker with the student's submission.
3. The expanded template is compiled and, in case of errors, these are reported to the student and the submission is aborted.
4. The JUnit tests are compiled in a separate step. Well written JUnit tests using the Java Reflection API should never cause compile errors.
5. The JUnit test itself is run using a *TestRunner* class that is slightly modified from the standard JUnit distribution. The only modification is that the test results are presented in a manner that is easier to parse by WeBWorK-JAG. Using a *.policy* file, the system is run in a very tight sandbox to reduce the risk to the server.
6. The score of the submission is the fraction of JUnit tests passed multiplied by the total number of points available for the problem. The student is shown his / her score for the attempt, as well as error messages describing failed JUnit tests.
7. In all cases (aborted or not), the temporary directory is removed.

6. WRITING WEBWORK QUESTIONS

Writing programming questions for automated assessment is hard and time consuming. In this section, we focus on the writing of WeBWorK-JAG problems. For the WeBWorK system, the process comprises the activities of specifying the question, the answer(s) and feedback, inputting them into WeBWorK using the PG language and, in the case of a WeBWorK-JAG question, writing JUnit tests. In this section, we will focus on WeBWorK-JAG questions, but some of the remarks can transposed to the case of multiple-choice and short answer questions.

Question Specification. While it is possible to write WeBWorK-JAG questions requiring students to write method snippets or several methods, we only focused on writing

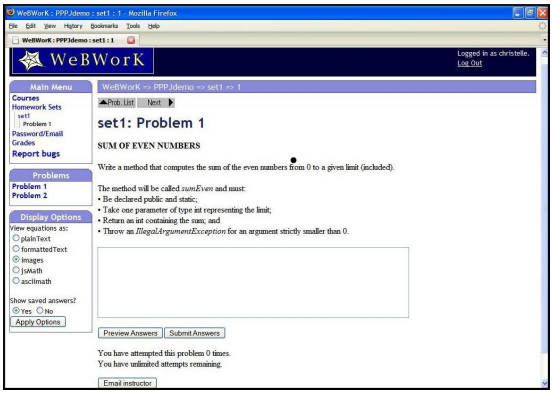

Figure 1: Screenshot of a WeBWorK-JAG Question for the Sum of Even Numbers Contributed by Students

questions that required students to write a unique and complete method. To simplify and standardize this, we designed a template to specify Java questions in a consistent way. The template is composed of the following:

- **Description** A short description of the method to be written.
- **Method name** The name of the method.
- **Method signature description** A description of the method signature in terms of:
  - Modifiers;
  - Type of the method, i.e. either static or instance method;
  - Number and types of parameters; and
  - Return type.
- **Exceptions** A description of the exceptions to be thrown along with the cases in which they are thrown.
- **Code** Code provided to support writing the method.
- **Notes** Particular restrictions concerning the method to be written.

An example of the instantiation of this template is provided below and illustrated in Figure 1.

Write a method that computes the sum of the even numbers from 0 to a given limit (included).
The method will be called *sumEven* and must:

- Be declared public and static;
- Take one parameter of type *int* representing the limit;
- Return an *int* containing the sum; and
- Throw an *IllegalArgumentException* for an argument strictly smaller than 0.

Writing Questions in PG and JUnit. Writing WeBWorK-JAG questions comes down to writing JUnit tests. JUnit is set up for expert users and, just like choosing compiler error messages, choosing test-failure messages is difficult but crucial to student feedback. Writing JUnit tests benefitted immensely from the use of the template. Good requirements are easier to test!

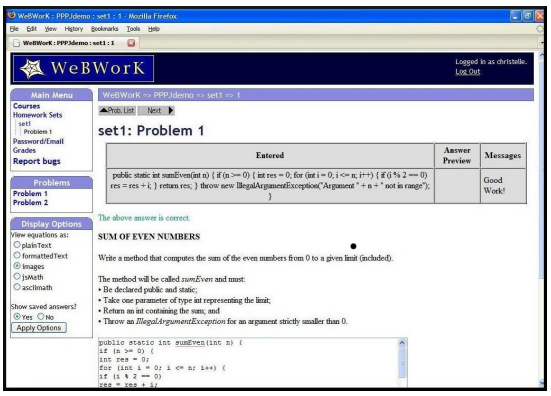

Figure 2: Example of Student Submission and Feedback for a WeBWorK-JAG Question - Right Answer

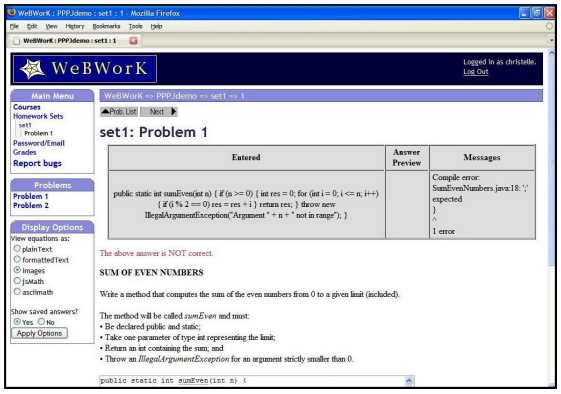

Figure 3: Example of Student Submission and Feedback for a WeBWorK Multiple-choice Question - Compilation Error

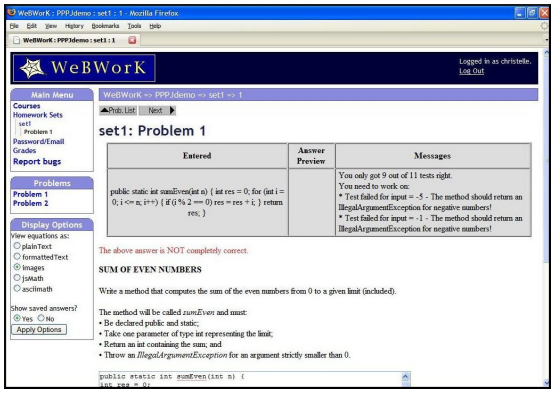

Figure 4: Example of Student Submission and Feedback for a WeBWorK-JAG Question - Wrong Answer

Quality Assurance. Quality Assurance requires checking the question specifications for comprehensibility and completeness, testing the WeBWorK-JAG questions by submitting method code with different types of mistakes, and evaluating and fine-tuning the effectiveness of the feedback. All exceptions of the input code must be caught and related error messages have to be generated. For example, non-excepted exceptions generated by input code (e.g. *NullPointerException* and *IndexOutOfRangeException*) must be caught to make the test fail. The granularity of the error messages is another important point to consider and is crucial for students' learning experience. Sample student feedback is illustrated in Figures 2 to 4.

7. USE OF WEBWORK

WebWorK was used for homework assignments by nineteen students taking a second Java programming course, Fundamental Computer Science with Java II (CS2), following on from Fundamental Computer Science with Java I (CS1). The first Java course covers topics that include procedural programming constructs (e.g. loops, conditionals, methods), arrays, recursion, and searching and sorting in arrays of primitive types. The second Java course emphasizes Java as an object oriented language and covers Java 1.5 features (autoboxing/unboxing, varargs, enum, generics, covariant return types and enhanced loops), memory management and data structures such as linked lists, stacks, queues, graphs and trees. In the CS2 course, students reviewed loops, arrays and recursion topics in two graded WeBWorK-based homework assignments, one composed of multiple-choice and short answer type questions, and the other composed of WeBWorK-JAG questions. All the homeworks were set up with an unbounded number of possible attempts to encourage active learning and to give students re-testing opportunities.

7.1 Students' Results

Multiple-choice and Short Answer Questions. The multiple-choice and short answer type questions consisted of determining the number of times a loop was executed, tracing code fragments and completing code. Very few students got perfect scores. We noticed that lower scores and a higher number of attempts were correlated, and appeared for problems on recursion and other questions with a large number of possible answers (such as multiple choice questions with multiple answers). 95% [1] of the students attempted a quiz composed of twenty questions with five possible answers more than eighteen times. 95% of the students attempted a question on tracing a recursive function and stating its argument calls more than nine times.

WeBWorK-JAG Questions. The WeBWorK-JAG problems consisted of simple questions such as an assignment problem, determining the negation of a Boolean value, whether a number is odd or not, and computing the area of a rectangle, and more complex questions involving sorting and recursion. The average number of attempts per WeBWorK-JAG question was twenty. Not surprisingly, problems linked

[1]The numbers in this paper are provided by the *Statistics* option in WeBWorK.

with sorting and recursion had the highest number of attempts. Here also, very few students got perfect scores and there is a correlation between the scores and the number of attempts. The high number of attempts for the simplest problems came from the fact that students did these problems first and needed to get used to the system in terms of understanding the way questions were formulated, how to answer them and defining a strategy to tackle the problem. Nevertheless, the free-form nature of the anticipated answers for these questions resulted in more errors and more attempts in general. Over time, most of the students corrected their errors and got the questions correct.

7.2 Students' Feedback

Specification and Presentation of the Questions. Programming question specification requires a large amount of setup information to be presented to the student, including class, method and variable names and pieces of code to be used in the solution. Due to the length and amount of information in the questions, students can miss the constraints demanded on the solution and, consequently, their number of attempts to solve a problem is increased. A long question necessitates the students to scroll down in the web browser, restricting their visibility and relying on short-term memory. Very often, students were found to print the questions and work on paper before submitting their work. Students cited a preference for having one question with one answer per page (or a single viewable screen). For WeBWorK-JAG questions, students asked to have an adequately sized text box for answers to permit them to read the code of previous attempts.

IDE-like Environment. Students were familiar with programming in the Eclipse IDE. When answering questions in WeBWorK however, they were expected to input code into a text box area that did not provide the efficient features of the Eclipse IDE editor, such as line numbering, the static compilation of code, syntax highlighting, code completion and access to the Java API help. Consequently, trivial syntax mistakes and typos were not detected before submission, resulting in multiple attempts as these became eliminated. Most of the students adopted a strategy to answer programming questions (and even to fill-in the blank and multiple-choice questions) that involved them tackling the programming questions in Eclipse first and then copying / pasting their solutions into WeBWorK for assessment purposes. This indicates a potentially necessary direction for the future development and use of WeBWorK within Computer Science linked with Eclipse.

Lack of Feedback in Multiple-Choice and Short Answer Questions. Students were very frustrated by questions composed of several sub-questions because the number of possible answers could be very high. For a quiz composed of twenty multiple-choice questions of five possible answers, the number of possible answers is extremely large. This meant that guessing the correct answer was impossible. The concern of the students was that the only feedback provided by WeBWorK in that case was *"At least one of the answers below is incorrect"*, somewhat akin to the familiar and highly instructional "syntax error" error message; WeBWorK did not pinpoint where students were wrong, creating an effect inverse to the one expected where students began to answer randomly and guess. They had to go over the whole problem instead of focusing on their wrong answers, resulting in inefficient learning. Future extensions to WeBWorK to integrate course materials into the system to provide more helpful and directed feedback to address some of these issues is the subject of our on-going research.

Lack of Feedback in WeBWorK-JAG Problems. Students proposed that some new features related to feedback be added to the current version of WeBWorK-JAG. Compilation errors are not parsed so as to be accessible for Java beginners; the compiler errors are simply printed with line numbers that are not always relevant. Students are interested in receiving more granularity and precision for compilation errors. They also want the solutions of the programming questions to be posted. The latter is obviously a pedagogical decision made by the instructor more than a shortcoming of the WeBWorK system itself. Students, used to the Eclipse IDE and the JUnit plug-in, asked for red and green failure or success indication in WeBWorK-JAG. At this point, the feedback states in text only the number of tests that failed out of the total number of tests that were run without providing any visual feedback (e.g. you only got 1 out of 4 tests right).

The quality of the feedback for WeBWorK-JAG problems in terms of JUnit tests depends on the quality of the JUnit code and the granularity of its messages for test failure. Instructors experimented with different degrees of granularity on this feedback: number of tests that pass / fail, exact test data and data of the same equivalence class implying failure. Students preferred getting specific feedback concerning the tests that failed, including test data of the test that failed or similar test data, and hints on how to fix the problem.

Distrust and Interaction with Instructors. Students liked WeBWorK; it made it easier for them to learn and practice programming at their own pace. However, the students inherently distrusted the system more than they would a human being playing a tutoring role of this nature. This meant that they actively tried to find mistakes in the WeBWorK questions when they did not manage to obtain the right answers. This is possibly because there is no ability to question and interrogate the system, whereas it is quite possible to query an instructor who can kindly (or not so kindly) point out the error in the students' problem solving strategy. Of course, this assumes that good quality assurance has already been undertaken!

8. STUDENT CONTRIBUTIONS

A very innovative pedagogy has arisen through our experiences of extending WeBWorK and using it to support our programming fundamentals classes. Students are tasked to contribute to WeBWorK by extending the problem set library themselves. They do this by writing multiple-choice, short answer, and WeBWorK-JAG questions, and by peer reviewing and testing the new questions. After being re-

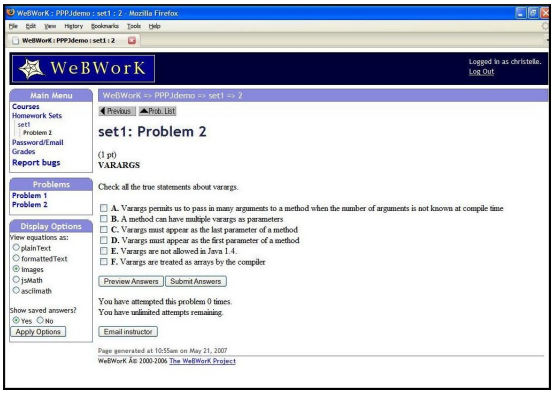

Figure 5: Screenshot of a WeBWorK Multiple-choice Question on Java Varargs Contributed by Students

worked by the instructors, the questions contributed by the students were integrated in the WeBWorK Computer Science library of problems. As part of their grade, students reflected on the difficulties and learning experience of undertaking this exercise.

Contributing Multiple-choice Questions. Students studied the features introduced in Java 1.5 and were asked to prepare short pieces of code that demonstrated their understanding of one of the features. They were tasked to contribute three multiple-choice or short answer questions to WeBWorK (See Figure 5 for an example).

The majority of the questions proposed by the students were straightforward and did not require a deep understanding of the feature they were supposed to showcase. However, formulation of the questions was often poor as students did not always take care to use the right terminology and many assumptions were often made. What is interesting is that they reported that they found it difficult to define relevant plausible but wrong possible answers for the multiple-choice questions.

Contributing WeBWorK-JAG Questions. Students also contributed to WeBWorK by specifying and writing JUnit tests for a programming question, based on the material covered in the CS2 class, to make accessible to their classmates. The JUnit tests are composed of tests that verify if the method is present, follows the signature description, is correct with respect to the specification and behaves correctly. This assignment helped students get familiar with the Java Reflection API and with unit testing.

The questions proposed by students included the sum of even numbers, checking whether a number is prime or not, a specific case of the binary search problem, testing if two arrays of ints are mirrors, a specific case of sorting an array of ints, two problems on linked lists (adding an element at the end or front of a linked list specified by its head and tail) and one question on counting the number of methods defined in a particular class using the Java Reflection API (Figure 1 shows an example contributed by students).

The main finding was the difficulty that students experienced in coming up with a question topic, defining a specific question and then evaluating its scope (i.e. required information). One of the interesting points is that students mentioned that, in the process of specifying the question, they always had the user in mind (i.e. students taking a similar course). They therefore tried to propose a question that had many different solutions and that would be considered challenging but feasible by their classmates. When initially asked to undertake this exercise, students were not provided with the template described in Section 6 (since it was not available at that time), but only provided with examples of questions previously written in WeBWorK. This certainly explains the poor question specification since the guide was found to later expedite the task. The modifiers and static / instance properties were not stated for example. Some students specified in the question that the expected answer should implement a particular algorithm (e.g. Euclid's algorithm to implement greatest common denominator, a recursive algorithm to implement merge sort, etc.), a requirement that could not easily be checked with JUnit. Some of the students, however, added related tests for these criteria. For example, in the sorting question, one student wrote code that ensured that *Arrays.sort* was not used and considered the use of that code fragment as cheating.

Overall, students stated that this assignment forced them to think about testing first and all the necessary steps they need to take to assure the quality of their question. They thought about the specification of the problem and the testing of the solution in parallel. The solution (code) was therefore not the main focus, which was the original objective we wanted to be reinforced by both the chosen system and the exercises we can undertake with it, as explained in Section 2. Once students were provided with sample WeBWorK-JAG problems, they had little problem writing tests for signature requirements, etc. Students also learned a great deal about the Java Reflection API.

The problems that arose were with writing the tests for the semantics of the method itself, rather than for the syntactical aspects of the code. Test coverage was one of the main issues here. Students very often forgot the boundary cases. When a method takes an object as a parameter, it is necessary to consider the case when that object is *null*; this case was too often forgotten. Students were not comfortable with identifying equivalence classes for the inputs of their method. They mentioned that writing the JUnit tests in the cases where exceptions were generated was particularly difficult. The documentation of JUnit covers valid inputs well but very rarely invalid inputs. Tests for invalid inputs were not written very well by the students. They tended to focus much more on back box testing rather than on white box testing, missing tests specific to the context of the method to be written. For example, for the binary search method, no tests were written related to finding an element in the middle of the array of ints. In the case of linked lists implemented by a head and a tail, no tests were written to check that the head and the tail pointers were well positioned when adding a node at the beginning and end of a linked list.

Another main issue was that students did not consider giving feedback to users which meant that they did not choose adequate test-failure messages (if any). So, while they had the user in mind when writing the question specification, they did not when it came to feedback, even though this was one of their main complaints when using WeBWorK-JAG. The experience of reviewing other's questions illustrated the important of and need for this though.

Peer Review and Bug Reporting. Students were asked to peer review the questions proposed by their classmates in terms of question specification and JUnit test cases by focusing on test coverage. In this way, questions and tests were improved before being submitted and integrated into WeBWorK by the instructors. Facilitating quality assurance, students tried out the set of multiple-choice questions in WeBWorK and submitted bugs to the instructors. Because of time constraints, the students did not attempt each other's WeBWorK-JAG questions.

WeBWorK Library of Questions for Computer Science. The problems written by the students were checked carefully by the instructors before being integrated into the WeBWorK library of questions for Computer Science. Multiple-choice questions were re-formulated and rewritten to be more challenging. WeBWorK-JAG questions were specified using the template described in Section 6. The JUnit code of the students had to be almost completely rewritten because of low coverage, missing test-failure messages and the incorrect treatment of exceptions (making the difference between exceptions generated by the user code with respect to *IllegalAssertionException* generated by JUnit). Specific test-failure messages (stating the data of the failed tests) were written to give students as much feedback as necessary when using the system.

Lessons Learned. Both the question formulation and wording, as well as the coverage of the solution, needs to be checked with student stakeholders; every question demands quality assurance and at least one pilot session with students to catch problems. The effort traditionally spent by an instructor assessing a submitted piece of work is now replaced with additional front end effort. Instead, the process of problem formulation and determination of success criteria activities is now substantially more difficult. Clearly, this is more efficient when amortized across multiple classes and large student numbers.

9. CONCLUSIONS AND FUTURE WORK

This paper has described WeBWorK as an alternative and open-source web-based system for the assessment of programming problems. Based on past successful experiences with adapting WeBWorK for use in supporting Computer Science programming fundamentals courses, we have extended the system to support the automatic grading of free-form Java questions by developing the WeBWorK-JAG. This paper explains this extension and illustrates how we have been using both the vanilla and additional capabilities of the WeBWorK environment to augment our courses. In particular, we have explained how WeBWorK provides instructors with a platform that can enable them to get students to contribute programming questions for assessment, an activity that reinforces the complementary programming skills of testing, problem formulation and quality assurance through peer review. At this stage, our findings are anecdotal, so controlled empirical studies of the effectiveness and impact of this style of pedagogy on learning what we consider to be the fundamental skills for the profession of Java programming have yet to be undertaken. We anticipate undertaking a larger and more systematic test this summer at Cornell College and at another institution. As mentioned earlier in the paper, we are also focusing on improving the interface and feedback mechanisms of WeBWorK.

One vision that we have for this system is the ability for communities of students to write problems for each other to share, practice and improve. One of the dominant issues with questions written within systems of this nature is trust – it is far more common for a student to think that the system is wrong than they are wrong when faced with an "incorrect answer" response. The nature of this issue makes it an ideal topic that can be addressed by community feedback and improvement.

Acknowledgments

This project is supported by NSF CCLI AI grant #0511385 and #0511391. We are grateful to Jacqueline Baldwin and Nathan Baur for their work on WeBWorK-JAG, Eileen Crupi, Tabitha Estrellado, Allyson Ortiz and Veronica Portas, our undergraduate assistants, for the input of WeBWorK problems and to John Lewis and William Loftus for permission to use some of their Java exercises from the book *Java Software Solutions: Foundations of Program Design (4th Edition), John Lewis and William Loftus, 2004*. We also thank our colleagues and Dean for their support.

10. REFERENCES

[1] J. Baldwin, E. Crupi, and T. Estrellado. WeBWorK for programming fundamentals. In *Proc. 11th Conference on Innovation and Technology in Computer Science Education*, page 361, New York, NY, 2006. ACM Press. (poster).

[2] J. Baldwin, E. Crupi, T. Estrellado, O. Gotel, R. Kline, C. Scharff, and A. Wildenberg. Examples of WeBWorK programming assignments. In *Proc. 37th SIGCSE Technical Symposium on Computer Science Education*, Houston, Texas, USA, 2006. (poster).

[3] K. Beck. *Extreme Programming Explained, Embrace Change*. Addison-Wesley, 2001.

[4] M. Bower. The effect of receiving the preferred form of online assessment feedback upon middle school mathematics students. In *Proc. Computers and Advanced Technology in Education*, pages 462–467, Kauai, Hawaii, USA, 2004.

[5] P. Brusilovsky and C. Higgins. Preface to the special issue on automated assessment of programming assignments. *Journal of Educational Resources in Computing*, 5(3), 2005.

[6] J. Cassady, J. Budenz-Anders, G. Pavlechko, and W. Mock. The effects of internet-based formative and summative assessment on test anxiety, perceptions of

threat, and achievement. In *Annual Meeting of the American Educational Research Association*, Seattle, USA, 2001.

[7] D.J. Charman and A. Elmes. *Computer Based Assessment 1: A Guide to Good Practice*. SEED Publications, Faculty of Science, University of Plymouth, UK, 1998.

[8] D.J. Charman and A. Elmes. *Computer Based Assessment 2: Case Studies in Science and Computing*. SEED Publications, Faculty of Science, University of Plymouth, UK, 1998.

[9] M. Covington and C. Omelich. Task-oriented versus competitive learning structures: Motivations and performance consequences. *Journal of Educational Psychology*, 76(6):1038–1050, 1984.

[10] M. Fagan. Advances in software inspections. *IEEE Transactions on Software Engineering*, 12(7), 1987.

[11] M. Gage, A. Pizer, and V. Roth. WeBWorK: An internet-based system for generating and delivering homework problems. In *Joint Meeting of the American Mathematical Society and the Mathematical Association of America*, 2001.

[12] M.E. Gage, A. Pizer, and V. Roth. Nsf 0088212 due ccli proposal. (obtained from Pizer, Gage and Roth).

[13] M.E. Gage and A.K. Pizer. WeBWorK – math homework on the web. In *Proceedings of the Annual International Conference on Technology in Collegiate Mathematics*, 1999.

[14] D. Galin. *Software Quality Assurance: From Theory to Implementation*. Addison-Wesley, 2003.

[15] O. Gotel and C. Scharff. Adapting an open-source web-based assessment system for the automated assessment of programming problems. In *IASTED Web-based Education Conference*, Chamonix, France, March 2007.

[16] Joint ACM/IEEE-CS Task Force on Computing Curricula 2001. Final report of the joint ACM/IEEE-CS task force on computing curricula 2001 for computer science, 2001. http://www.computer.org/education/cc2001/final/index.htm.

[17] Joint Task Force for Computing Curricula 2005. Computing Curricula: 2005 Overview Report. A cooperative project of the Association for Computing Machinery (ACM), The Association for Information Systems (AIS), The Computer Society (IEEE-CS), September 2005.

[18] C. Scharff and A. Wildenberg. Teaching discrete structures with SML. In *Proc. Functional and Declarative Programming in Education*, Pittsburgh, USA, 2002.

[19] C. Weibel and L. Hirsch. WeBWorK effectiveness in rutgers calculus, July 2002. http://math.rutgers.edu/~weibel/ww.html.