ADAPTING AN OPEN-SOURCE WEB-BASED ASSESSMENT SYSTEM FOR THE AUTOMATED ASSESSMENT OF PROGRAMMING PROBLEMS

Olly Gotel and Christelle Scharff
Pace University
Seidenberg School of Computer Science and Information Systems
New York, NY, USA
{ogotel, cscharff}@pace.edu

ABSTRACT
This paper describes the adaptation of an open-source web-based assessment system to support the teaching and assessment of programming fundamentals. WeBWorK is a system that has been used effectively in mathematics for a number of years and the nature of its underlying engine makes it particularly well-suited to the problems from this discipline. It is accessible for instructors to contribute problems and free for students to use. A community has built up around WeBWorK to share problem sets in mathematics. Within computer science, many of the web-based programming delivery and assessment systems are commercial initiatives; students pay to use them and problem sets are administered centrally. This paper describes a project that is adapting WeBWorK for use in computer science and presents initial findings from its use over one academic year. Our on-going and future work is summarized, and the broader potential highlighted.

KEY WORDS
Computer Science Education, Reusable Learning Objects, Teaching Programming Fundamentals, Testing and Assessment, Web-based Education, WeBWorK.

1. Introduction

Learning to program lies at the heart of computer science education at present. Rightly or wrongly, this is one of the first skills that a computer science undergraduate is expected to master. In recognition that this is a skill that takes time and practice to acquire, and that students need guidance and feedback beyond that which can be provided in the classroom, a number of web-based systems for the delivery and automatic assessment of programming assignments have been developed. A selection of these systems is described in Section 2. By using a web-based system that basically acts as a 'personalized programming tutor' to assess, reinforce and improve a student's understanding of programming, the conjecture is that students will obtain a stronger conceptual foundation for subsequent computer science courses. There has been promising research on the relationship between on-line feedback and academic performance [1,2,3,4], as well as on the role of re-testing opportunities in mastering learning [5].

This paper describes the early work of an initiative that attempts to enhance the learning experience of undergraduate students in computer science and establish a sound pedagogical environment to increase student achievement in programming. The initiative is unique in that it focuses on the use of an open-source system called WeBWorK [6,7] that is entirely extensible and free for use. It supports randomization due to its flexible problem authoring capability and promotes an underlying philosophy that relies on the 'community' to share and assure the quality of problem sets.

The overall objectives of this work are to: 1) engage students in active on-line learning to augment traditional practices with all the core computer science courses; 2) provide students with immediate and customized feedback on their progress; 3) provide students with tailored and constructive support for any problem areas they encounter; and 4) provide instructors with the ability to continually monitor and assess student performance (both individually and across classes), to help guide class sessions. This paper focuses mostly on the first objective.

The paper is organized as follows. Section 2 summarizes related work and provides the context for our work with WeBWorK. Section 3 describes the WeBWorK system and its use within mathematics. It explains how we have been adapting it to support programming problems. Section 4 outlines pilot studies we conducted to trial its use in programming classes. Section 5 provides a discussion of our findings and the wider implications for web-based problem delivery and assessment. Section 6 provides conclusions and a description of our future work.

2. Systems for Automated Assessment

A number of systems for the delivery and automated assessment of programming assignments have been developed, most of which are web-based and include increasingly advanced testing approaches. A selection can be found in a journal issue dedicated to the topic [8].

Problems supported range from true / false, multiple-choice and matching problems 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. QuizPACK (*http://www2.sis.pitt.edu/~taler/QuizPACK. html*) is a free system that generates parameterized matching questions focusing on tracing C programs / fragments. Proprietary web-based systems, such as CourseMarker (*http://www.cs.nott.ac.uk/CourseMarker*), CodeLab (*http://www.turingscraft.com*) and MyCodeMate (*http://www.mycodemate.com*), focus on Java and C++. 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. A recent enhancement is based on customizing the problems to individual students. CodeLab delivers only short answer exercises, focusing on a particular programming construct or idea. MyCodeMate permits to compile and run code but does not check correctness. CourseMarker and CodeLab problems may be selected from fixed (purchased) databases – instructors may not customize problems, and they have per-user costs. 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. 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. MyCodeMate is the only system offering graphical-based progress and performance visualization. (CFX (*http://cfx.sourceforge.net*), Marmoset (*http://marmoset.cs.umd.edu*), Viope (*http://www.viope.com*), OWL (*http://owl.course.com*) and TRAKLA2 (*http://www.cs.hut.fi/Research/TRAKLA2*) are other systems.

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 parameterization and mathematical formulae symbolic treatment) could address a wider range of problems associated with programming.

3. WeBWorK

3.1 WeBWorK for Mathematics

WeBWorK (*http://webwork.math.rochester.edu*) [6,7] is a free open-source web-based formative assessment system to generate, deliver, and (automatically) grade homework problems and distribute their solutions. It is used by over 50 institutions worldwide to teach mathematics (calculus, pre-calculus, algebra, applied, finite and discrete mathematics). 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," and to encourage active learning [7].

Instructors can select and edit problems from a problem library or write their own customized WeBWorK problem sets (with their solutions, grading schemas and deadlines) using the Problem Generating (PG) 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. Problems that can be defined in WeBWorK range from true / false and multiple-choice problems to sophisticated matching problems (evaluated via scripts). Initially designed to deliver mathematics problems, WeBWorK, through PG, has the ability to recognize and substitute formulae (e.g. WeBWorK would equally accept x+1, (x^2-1)/(x-1) or x+sin(x)^2+cos(x)^2 as an answer), and to generate individualized versions of problems using predefined pseudo-random and Perl functions. With respect to multiple-choice questions, not only does WeBWorK allow questions to be presented in random order, within each question the possible answers can be presented in random order. In addition, parameterization of the question itself means that the specifics of the question can be random. The result is that no student should get the same set of problems.

Instructors have access to statistics that provide "information on the performance of individual students and the course as a whole," and the system permits the monitoring of students' work [7]. A study at Rutgers State University showed a strong correlation between WeBWorK scores and final scores in calculus, especially for first-year students [9]. Similar results were found at Stony Brook University for discrete mathematics [10].

3.2 WeBWorK for Programming Fundamentals

In the fall 2005, we began to adapt and extend WeBWorK for use in the core courses of the computer science curriculum, referred to in "Computer Science Curriculum 2001" as *Programming Fundamentals* [11]. These core courses revolve around the following computer science essentials: fundamental programming constructs, algorithms and problem-solving, elementary data structures, recursion and event-driven programming. Our WeBWorK server is accessible for full experimentation at *http:/198.105.44.187/webwork2/demo.*

We developed WeBWorK problems written in PG for the Java, Python and SML (Standard Meta Language) programming languages [10,12,13]; Java is commonly used to teach computer science students, Python is becoming widely accepted by programmers and SML is introduced when students learn about functional programming. Based on the pre-existing paper-based problem sets of the instructors, we reformulated these as simple true / false, multiple-choice and parameterized matching problems that would be amenable to WeBWorK. These problems were designed to test students on terminology, basic programming concepts, and the syntax and semantics of Java, Python and SML. Permission was also granted by the authors of a leading textbook to use sample student questions (*Java Software Solutions: Foundations of Program Design (4th Edition), John Lewis and William Loftus, 2004*). Whenever possible, the PG was written so that problems would be randomized and individualized on a per-student basis.

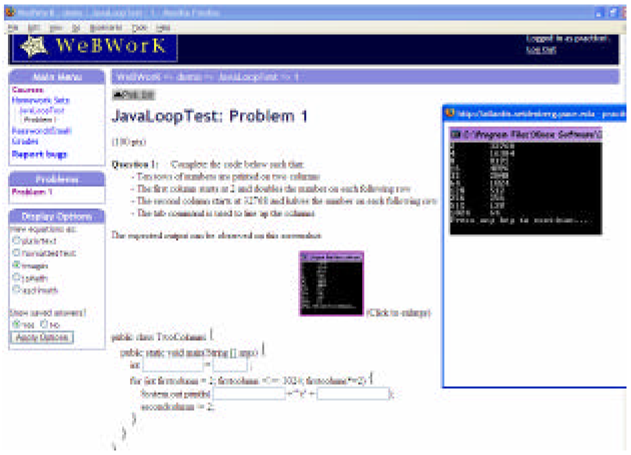
Examples of simple problems are as follows:

- Assuming *X* and *Y* are integers, both randomized and individualized, what is the value of *X % Y* in Java?
- Assuming the value *X* is randomized and individualized, what is the highest number printed by the statement *for counter in range(0, X, 1): print counter* in Python?
- In SML, assuming *f* is a function of type *X ? X*, where *X* is a randomized and individualized type, what is the type of *map f L* when *L* is a list of elements of type *X*?

We also wrote more sophisticated WeBWorK problems that require use of the complete features of PG to assess that students can apply the learned concepts (see Figure 1). We defined macros to be able to write Java, Python and SML problems more easily with a standard coding style. Correct indentation was particularly important for Python where code blocks are defined by their indentation levels. Characters commonly used in programs, such as quotes for strings and underscores in the names of variables or functions / methods, are special characters in LaTeX, one of the sub-languages of PG, and needed to be redefined as macros. However, for WeBWorK to truly have a significant impact in those courses spanning the programming fundamentals, it needs to be able to evaluate programs. Colleagues at Cornell College have started extending WeBWorK to provide it with the ability of evaluating Java program / fragment correctness by interfacing it with JUnit (*http://www.junit.org*), an open-source testing framework. This extension will increase the flexibility in answering programming questions.

4. Pilot Studies

We planned to use WeBWorK in two different instructional settings over the course of the 2005-2006 academic year: to deliver online homework assignments and to deliver in-class tests. WeBWorK has been primarily used for homework assignments in mathematics and we wanted to understand the distinction between delivering and assessing homework assignments at home versus in a test-based setting in the classroom, more so with respect to student acceptance in the first instance rather than with respect to the impact on learning. The research objective was thus to determine whether a system primarily used for at-home personal study could also be used to administer personalized in-class tests.



JavaLoopTest: Problem 1

Question 1: Complete the code below such that:
- Two rows of numbers are printed on two columns
- The first column starts at 2 and doubles the number on each following row
- The second column starts at 32768 and halves the number on each following row
- The tab command is used to line up the columns

The expected output can be observed on this screenshot.

```
public class TwoColumns {
  public static void main(String [] args) {
    int ______ = ______;
    for (int firstcolumn = 2; firstcolumn <= 1024; firstcolumn*=2) {
      System.out.println(______ + "\t" + ______);
      secondcolumn /= 2;
    }
  }
}
```

Figure 1 – Screenshot of a WeBWorK Loop Problem in Java

We planned to target two audiences in our pilot studies: (a) freshman computer science students / students undertaking a first course in programming who were not necessarily freshman; and (b) junior / senior computer science students. The research objective was to examine whether maturity and prior knowledge was an important factor in motivating the use of the system.

WeBWorK was used in CS121 (Programming I) and IS223 (Fundamentals of Programming) in the fall 2005 and in the spring 2006 to deliver Python and Java in-class tests (30 to 60 minutes) and homework assignments. CS121 is the first core course of the computer science curriculum; it is taken by freshmen. IS223 is taken by information systems students of all levels as their compulsory introduction to programming concepts. 21 students participated in the fall and 24 students participated in the spring. When in-class tests were taken, control groups were established to compare the feedback of students exposed to the different approaches. Four tests were held in each course and the test set-up varied each time based on our learning. WeBWorK was also used in CS361 (Programming Languages and Implementation) in the fall to deliver SML homework assignments. CS361 is a core course of the computer science curriculum taken by junior / senior students. 12 students participated.

We gathered data regarding students' perceptions of WeBWorK with regard to its benefits (or not) when used for homework assignments and for in-class tests. We also collected data on the students' patterns of use (i.e. whether used, number of attempts at questions, etc.). While the findings relating to student perceptions are not statistical, we found that WeBWorK did seem to be better suited to homework assignments rather than for in-class tests. These findings are discussed in Section 5. However, important issues were found associated with its use for homework assignments also, particularly with respect to use by students new to or less confident in programming. These issues are also highlighted in Section 5.

The instructors who used WeBWorK shared their experience of designing WeBWorK problems, the way in which students approached WeBWorK use, how WeBWorK affected the way they taught their courses and its impact on class interactions. The use of WeBWorK on student performance, and how it helps students learn programming fundamentals, is still to be studied in more detail. Tentative statistical data was gathered but many user interface and logistical issues were found to require attention before this could be measured independently.

5. Findings and Lessons Learned

5.1 Student Perspective

Test presentation. Within WeBWorK, problems can be presented individually on a screen or in a list on a shared screen. For in-class tests, students were initially provided with 20 multiple-choice / matching problems in one browser page and they could submit their test answers by pushing the "submit" button only once. This approach was not found to offer students any suitable visibility of progress on the test, as questions can scroll beyond a viewable screen, and too final. It was not found as accessible as a paper booklet and did not lend itself to the activity of answering questions in the order students wanted. When taking paper-based tests, students may annotate the test booklet and mark questions to return to later. Electronically, there are no visible cues to help do this. Consequently, many students submitted tests with questions entirely missed out in the early stages of the pilot. The alternative, to place single questions on a screen, was found equally problematic to navigate from a master list page. In addition, WeBWorK does not offer a flexible marking system; it does not permit to deduct marks for wrong answers to eliminate guessing. We thus experimented with on-line test presentation and layout during the year and have yet to find an acceptable way.

Test taking. Initially, some students were not comfortable that they did not have a hard copy of the test where they could draft and work out their answers. This is particularly important when students like to trace through changing variable values and loop iterations. We thus experimented with different combinations of test material: electronic only, electronic and accompanying hardcopy, and hardcopy only, so as to compare both the differences in test performance and perception of the test when administered in different ways. We found no correlation with performance, but students taking the paper-based tests were less anxious. Another issue we faced was the size of blank boxes where answers are expected for matching questions (as shown in Figure 1). The dimensions of the box were found to be leading with respect to likely answers and so needed standardization.

On-line test priming. During our experiments with test taking, we converged on giving students a set of programming questions and time to tackle them initially using their regular coding environment (Eclipse). We then moved the students into the WeBWorK environment where they were exposed to a set of multiple choice and / or matching questions based upon the given task they had just attempted. The intention was for students to transfer their immediate prior experience to help in answering the WeBWorK questions, indicative of deeper learning. This blended a mixture of approaches, confused the weaker students but had the desired effect with stronger students.

Unexpected solutions. There are many ways students can attempt to solve the same programming problem and support for this variability needs to be factored into any successful on-line programming assistant. Some form of code-priming seems to be desirable before going into a question / answer / feedback electronic environment if this support is not at the compilation level.

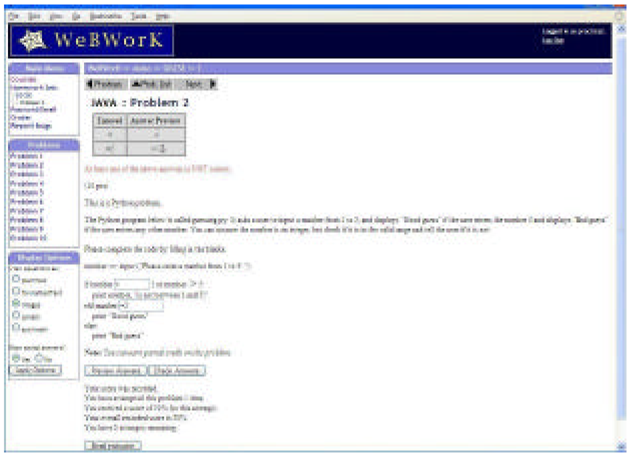


Figure 2 – Example of Student Submission

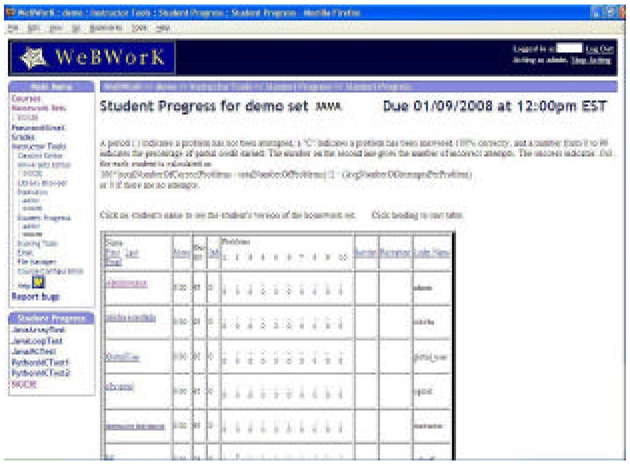
Feedback. When used for homework, WeBWorK was set up in such a way that students could attempt the same problem an unlimited number of times. Students liked the freedom of access offered by WeBWorK; they could access it from anywhere and do only a subset of the required problems and so practice at their pace. They liked instant feedback and self-assessment at home. However, the feedback provided by WeBWorK consists of only specifying whether an answer is right or wrong (see Figure 2). While instant feedback of this nature was considered an advantage for homework, where multiple attempts were allowed, it was considered a disadvantage for in-class tests. Some students were shocked to see results at the click of a button and could not concentrate for the remainder of the class. A correct / incorrect response is coarse grain and not constructive. The feedback mechanism within WeBWorK has limitations and could be damaging from a pedagogical standpoint for some students. Approaches to integrate more constructive feedback is the subject of our on-going research.

5.2 Faculty Perspective

Crafting WeBWorK-ready questions. Instructors found that designing WeBWorK homework assignments and tests required significantly more time than designing paper-based ones. Additional care had to be given to find the right formulation for a question, where only a limited number of answers are expected. The average time to write and test a seemingly trivial question can be a few hours. To compound this issue, WeBWorK can support complete problem randomization (in both the questions and answers) such that students get individualized problems. This feature has the advantage of reducing plagiarism and cheating, but it also requires more quality assurance (QA); instructors must QA several versions of the same problem. The interesting point is that, once done, it appears to be quite straightforward to change the questions to other programming languages; for example, the problems written for Python could be easily rewritten to target Java. However, the quantity of testing required is an issue that needs further study and possibly some automation. Not accepting a correct student answer in an in-class test can be very disrupting. Likewise, not accepting an ingenious solution to a problem during home use of WeBWorK could easily deter students, ultimately reducing their perceived value of the system.

Balance of administrative effort. WeBWorK questions take a long time to craft and test, but this is offset by the speed with which a student's attempt at a question can be assessed and the reuse potential (assuming the QA has been performed sufficiently). Figure 3 shows the current mechanism for instructor feedback in WeBWorK and this is something that is being re-examined in our on-going work. However, there are obvious difficulties in persuading faculty to spend the time upfront creating and testing questions. As a library of reusable, parameterized and pre-tested questions evolves, this issue becomes less evident. There is also a logistical task associated with creating and maintaining student accounts, and with preparing the test environment (selecting questions, arranging for start times and end times, and selecting the manner in which to display feedback, as shown in Figure 4) This can be simplified by student assistance, but there is a need to examine ways in which to ease the overhead on individual instructors to encourage uptake.

Motivating students to use WeBWorK. Instructors observed a difference in the way freshman and junior / senior computer science students approached WeBWorK for on-line homework assignments. Very few freshman students accessed the WeBWorK homework assignments even if the assignments were part of the final grade for the course. The students who accessed the assignments were either the best students in the class or students with difficulties, who proceeded by answer elimination to get the right answers. All junior / senior students accessed the WeBWorK homework assignments and did not proceed by answer elimination (except one). This difference in behavior among freshman and upper-level students may be due to the different levels of maturity, pre-existing familiarity with coding and their commitment to study computer science, and is an important factor.


WeBWorK
Student Progress for demo set JAVA
Due 01/09/2008 at 12:00pm EST
Report bugs
Figure 3 – Instructor View: Student Progress

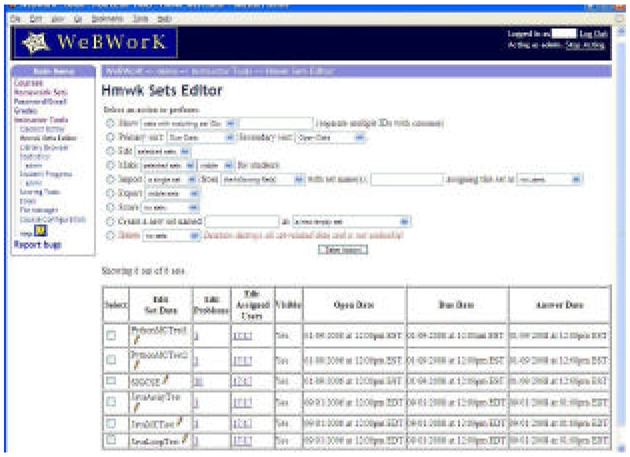


Figure 4 – Instructor View: Homework Sets Editor

Impact on regular class sessions. Instructors agreed on the fact that WeBWorK did not change the way they taught the material in their courses; rather, WeBWorK was used as an add-on and integrated in the courses in a smooth manner for additional homework assignments and in-class tests. However, those class sessions that used WeBWorK for the tests did get somewhat overtaken with the use of the system rather than the material being covered. Technical problems often meant that the tests took longer than planned (e.g. problems with Internet access; server clock not being synchronized and timing students out of tests early and losing their answers). Many of these technical issues were gradually eliminated during the course of the year, but are worth noting when starting out with in-class web-based educational assistance.

Randomization. This feature proved a good way to mitigate plagiarism in in-class tests and (possibly) in homework. However, when experimenting with test delivery to see which option worked best for the students it was found logistically very difficult to deal with pre-printing individual test papers to align with a student's variant of a test. This is not a viable way forward as it increases the instructor's pre-class work considerably.

Class feedback. The randomization feature also led to changes in the instructor interaction with the class. This feature makes it very difficult for instructors to go over the solutions of homework assignments and tests in class since instructors need to choose one version to review. If students can transfer their understanding between their problem and its variants it would demonstrate a real understanding of the concepts. However, this has not always proven achievable in class sessions and can lead to long and lengthy explanations of every unique case.

6. Conclusions and Future Work

As technical issues were overcome during the pilots with using WeBWorK for computer science, the issues with question formulation, the large possible answer space and the amount of QA / testing became prominent. As these were addressed, the main barriers were found to be presentation and feedback at the student level, and administration and feedback at the instructor level. We did not focus on the mechanisms WeBWorK uses to convey student and class performance during the pilots (e.g. alerts to common class-wide problems) because, essentially, the user interface was not found to support many tasks as well as it could. Progress and performance visualization (for students and faculty) is important for uptake and the emphasis of our current research.

Pertaining to the research questions of the pilot studies, while the use of WeBWorK for in-class tests did not receive favorable feedback in comparison to its use for homework, many of the issues leading to this impression were a direct result of the constraints imposed by the interface, not limitations in task support per se. Student maturity, familiarity with the broad topic of study, confidence level and motivation to learn were found to be the key factors driving use of WeBWorK at home, irrespective of whether use was officially graded or not.

Our long-term goal is for WeBWorK adaptation is to detect an individual student's learning needs based on their attempted answer and present simpler questions and / or prerequisite tutorial material to promote a tailored learning program, rather than presenting just wrong or right feedback. We are currently investigating such improvements in the overall "WeBWorK experience".

The randomization possibilities provided by WeBWorK makes it extremely powerful and capable of providing students with a unique learning experience. Coupled with targeted instruction based on needs, the system could be a valuable contribution to computer science education. An open-source system of this kind requires a community of contributors to monitor quality and share work. This has been achieved in mathematics and we are working to do likewise in computer science. A freely available web-based assessment system for instructors and students may be far more broadly applicable in education.

Acknowledgements

This project is supported by NSF CCLI AI grant # 0511385. We are grateful to our undergraduate assistants for the input of WeBWorK problems and to Lewis and Loftus for permission to use some of their Java exercises. We also thank our colleagues and Dean for their support.

References

[1] M. Bower, The Effect of Receiving the Preferred Form of Online Assessment Feedback Upon Middle School Mathematics Students, *Proc. Computers and Advanced Technology in Education*, Kauai, Hawaii, USA, 2004, 462-467.
[2] J. Cassady, J. Budenz-Anders, G. Pavlechko, & W. Mock, The Effects of Internet-Based Formative and Summative Assessment on Test Anxiety, Perceptions of Threat, and Achievement, *Annual Meeting of the American Educational Research Association*, Seattle, USA, 2001.
[3] D.J. Charman, & A. Elmes, *Computer Based Assessment 1: A Guide to Good Practice* (SEED Publications, Faculty of Science, University of Plymouth, UK, 1998).
[4] D.J. Charman, & A. Elmes, *Computer Based Assessment 2*: *Case Studies in Science and Computing* (SEED Publications, Faculty of Science, University of Plymouth, UK, 1998).
[5] M. Covington, & C. Omelich, Task-oriented versus Competitive Learning Structures: Motivations and Performance Consequences, *Journal of Educational Psychology*, *76*(6), 1984, 1038-1050.
[6] M. Gage, A. Pizer, & V. Roth, WeBWorK: An Internet-based System for Generating and Delivering Homework Problems, *Joint Meeting of the American Mathematical Society and the Mathematical Association of America*, 2001.
[7] M. Gage, A. Pizer, & V. Roth, *NSF 0088212 DUE CCLI Proposal*, 2000 (obtained from Pizer & Gage).
[8] P. Brusilovsky, & C. Higgins, Preface to the Special Issue on Automated Assessment of Programming Assignments, *Journal of Educational Resources in Computing*, *5*(3), 2005, 1.
[9] C. Weibel, & L. Hirsch, WeBWorK Effectiveness in Rutgers Calculus, July 2002 (*http://math.rutgers.edu/~weibel/ww.html*).
[10] C. Scharff, & A. Wildenberg, Teaching Discrete Structures with SML, *Proc. Functional and Declarative Programming in Education*, Pittsburgh, USA, 2002.
[11] Final Report of the Joint ACM/IEEE-CS Task Force on Computing Curricula 2001 for Computer Science.
[12] J. Baldwin, E. Crupi, & T. Estrellado, WeBWorK for Programming Fundamentals, *Proc. 11th Conference on Innovation and Technology in Computer Science Education*, ACM Press, New York, NY, 2006, 361-361.
[13] J. Baldwin, E. Crupi, T. Estrellado, O. Gotel, R. Kline, C. Scharff, & A. Wildenberg, Examples of WeBWorK Programming Assignments, *Proc. 37th SIGCSE Technical Symposium on Computer Science Education*, Houston, Texas, USA, 2006.