IMPROVING AUTOMATIC CODE ASSESSMENT

Brad Vander Zanden and Michael W. Berry
Electrical Engineering and Computer Science Department
University of Tennessee
Knoxville, TN 37996
865 974-8175
bvz@eecs.utk.edu, berry@eecs.utk.edu

ABSTRACT
In Fall Semester 2012 we started using an automatic code assessment system named CodeAssessor to grade students' coding problems on exams. This paper describes some of the lessons we have learned in using automatic code assessment and the improvements to CodeAssessor we have made as a result of them. The students' results on the exams have improved from a median of 8/25 on the first exam given in Fall Semester 2012 to a median of 19/25 on the first exam given in Spring Semester 2013.

1 INTRODUCTION

For years we have given students in our CS1 course exams that test a combination of the students' understanding of concepts and their programming skills. The programming skills segment of the exams involves handwriting code fragments that are then graded by graduate student teaching assistants. This approach has a number of drawbacks:

1. The grading can vary considerably amongst TAs.
2. Students have 75 minutes to complete both the coding and concept portions of the exam, so we have to ask rather simple coding questions.
3. Students frequently have to erase and re-arrange portions of their answers, which leads to very messy answers that can be difficult to grade.
4. Students frequently have logic errors that they do not catch because they are unable to test their programs on sample input.

For all these reasons we looked for a computerized coding environment that would provide automatic grading. We did not find any such tool and hence devised the CodeAssessor tool, whose initial design was initially reported on in [1]. CodeAssessor is a form-based, web tool that allows us to provide parts of a program for the student and allows the student to fill in code blocks to complete the rest of the program. The student can compile and test the program, submit it for grading, and can ultimately view the instructor's model solution. The grading component of CodeAssessor compares the student's output on test cases against the instructor's output and awards credit when the output matches on all the test cases. Students receive an instructor-specified number of attempts for each code block, and an instructor-specified point deduction for each failed attempt. If the student fails to correctly write a code block in the allotted number of attempts, or is stuck and

needs to move on, they are shown the solution. Hence students receive immediate feedback as they progress through problems, and the feedback also ensures that they do not get stuck for prolonged periods of time. Such immediate feedback has been shown to be useful in other interactive tools [2,4,5]. CodeAssessor currently supports C and C++ programs, but a couple line change to its PHP backend would allow it to be easily adapted for other languages, such as Java.

In Fall 2012 we started using CodeAssessor for the coding question portion of our CS1 exams. In this paper we describe our experience with using CodeAssessor for performing automatic code assessment for exams during Fall Semester 2012, how we were able to improve results as we proceeded through the semester by modifying aspects of CodeAssessor, and finally initial results from Spring Semester 2013 after further tweaks to the Code Assessor interface over winter break 2012-2013. The changes resulted in students improving from a median of 8/25 on the first CodeAssessor exam in fall semester 2012 to a median of 19/25 on the first CodeAssessor exam in spring semester 2013.

The rest of this paper is organized as follows. We first describe related work and then describe both the initial student interface for CodeAssessor and the current interface that has evolved from both observation of students and feedback from students. We next discuss additional lessons we have learned in creating problems for automatic assessment and finally we summarize the improvements in student scores.

2 RELATED WORK

We are not aware of any automated code assessment systems that are used for exam purposes in the CS literature. There are a number of existing automated problem-solving environments that are targeted at introductory computer science students, and that provide practice with drill work on targeted parts of a programming language. These online tutors include MyProgrammingLab [6], Practice-It!; Practice-It! [7], and Problets [2,3]. All three tutors provide students with a mixture of questions that give them practice with computer syntax (e.g., which of the following ids are valid variable names), code reading (e.g., what is the output of the following code), and programming (e.g., write a for loop to compute and output the first 10 Fibonacci numbers). Many of the problems allow the students to make multiple attempts and report whether or not the student successfully completed the problem. For more complicated problems, the Problets tutor also provides feedback during the problem that helps guide the student toward a solution of the remainder of the problem. None of these systems however are oriented toward having a student write parts of an entire program or for grading the student's efforts. Practice-It!, Practice-It! comes closest to our system in terms of the way it operates on programming fragments. Like CodeAssessor, it runs both an instructor's solution and the student's solution against a set of test cases, and uses a "diff" command to compare the two sets of outputs. Also like

CodeAssessor, MyProgrammingLab and Practice-It!, Practice-It! have tools that allow instructors to create their own problems.

3 EVOLUTION OF THE STUDENT INTERFACE

CodeAssessor has a student view that allows students to complete the problems and an instructor view that allows the instructor to create and publish problems. This paper focuses on the student view and the improvements made to the student view as a result of our experience with using CodeAssessor for exam testing. In this section we give a brief overview of the initial student interface and then discuss how we improved it based on our own observations and student feedback.

3.1 Student View

Figure 1 shows a sample screen shot of the student interface for CodeAssessor as it was first presented to students in Fall Semester 2012. The web page begins with a brief description of the problem, and then alternates between code blocks provided by the instructor and code blocks that the student is expected to complete. Student code blocks are prefaced with a set of instructions, the number of points available for that code block, and the deduction for each attempt with that code block. At the bottom of the screen is an area for the student to enter his or her own test cases. At the top of the screen are three buttons that allow the student to save their work, compile and test their program, and submit their program for grading.

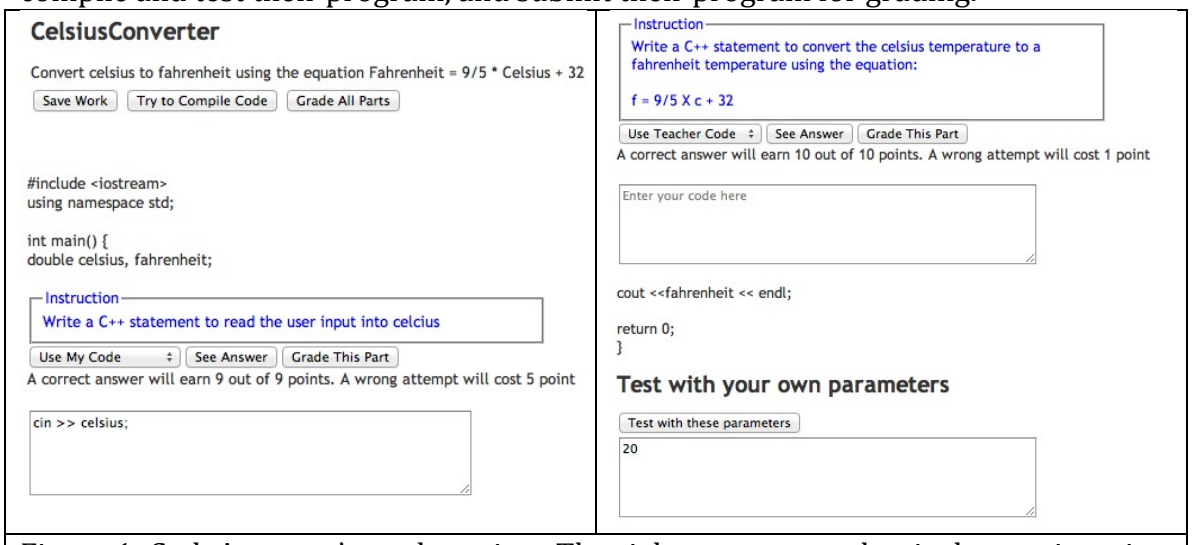

Figure 1: CodeAssessor's student view. The rightmost screenshot is the continuation of the web page shown in the leftmost screenshot.

Rather than use the "Grade All Parts" button, the students were encouraged to incrementally develop their program by filling in one code block at a time, and then selecting the "Grade This Part" button. The remaining code blocks are filled in with the instructor's solution (note that the "Use Teacher Code" item has been selected for the second code block in Figure 1). The "Show Answer" button associated with each code block can be pressed at any time. If the student has successfully

completed the code block or exhausted the points available for that code block, the student is shown the answer. If the student has not yet successfully completed the code block and has points available, the student is warned that seeing the answer will result in them being awarded 0 points for this code block, and asked if they still wish to see the solution.

3.2 Fall 12 Semester Revisions to the Student View

It quickly became apparent that allowing students to grade all the parts was counterproductive, because we were trying to encourage incremental development and testing of programs, and the grade/compile all parts buttons allowed students to follow their natural inclination, which is to write the entire program and then start testing it. Unfortunately, this type of development tends to lengthen the time required to debug a program, which is not a desirable characteristic in the time-limited, pressure-laden crucible of an exam. Hence after the first exam we eliminated the Grade All Parts options from the top of the interface, and instead forced the student to use the "Grade This Part" button.

3.3 Inter-Semester Revisions to the Student View

At the end of Fall Semester 2012 we asked students to tell us what they would like to see improved about Code Assessor. Based on these comments, a number of changes were made to Code Assessor between Fall Semester 2012 and Spring Semester 2013. The updated Code Assessor interface is shown in Figure 2. The important changes included:

1. Numbering the lines in each code block and editing the output from the compiler so that the line number errors reported by the compiler

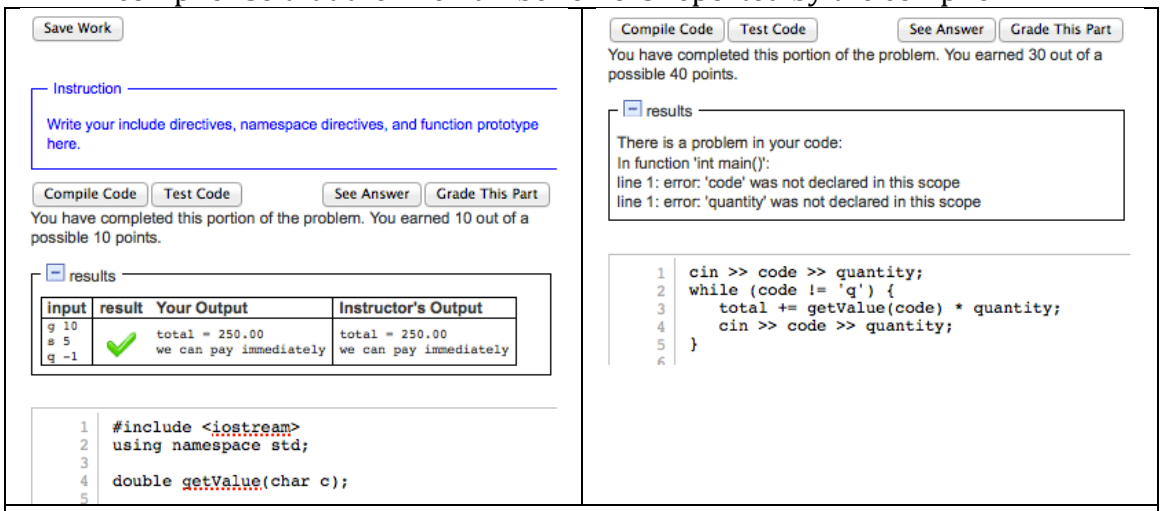

Figure 2: Updated CodeAssessor interface with numbered code blocks, and compiler messages, test results, and grade results grouped next to the code block the student is currently working on. The rightmost screenshot is the continuation of the web page shown in the leftmost screenshot.

corresponded to the line numbers in the code blocks. For example, if an error occurred at line 43 in the entire program, but at line 5 in the code block, then the error message was edited to say that the error occurred at line 5.

1. Modifying the buttons associated with each code block. We added a compile button to each code block so that students did not need to go to the top of the form to compile their code block. We also added a "Test Code" button to each code block so that students could enter their test cases immediately above the code block and run a test case, rather than having to go to the bottom of the web page to test their program. When we added these modifications, we were able to eliminate the "Use My Code" button, since it is always apparent when the user selects the "Compile", "Test", or "Grade" buttons which code block they want used.
2. Changing the "Test your Program" form so that the student's test data was run against both the student's program and the instructor's program, and the output from both programs was shown side-by-side. Additionally, a green check mark or red x was added to show whether the student's output matched the instructor's output.
3. Adding collapsible blocks above each code block that are shared by the compile, grade, and test code buttons and which provide 4 types of information:
    a. compiler error messages
    b. a form that lets the user enter some test input and then run their program
    c. the student's and instructor's output for test input
    d. the student's and instructor's output for each of the test cases used for grading.

    Previously error messages had been shown in a pop-up dialog box that disappeared when the user selected "ok". Students understandably wanted to go back and reference this information as they modified their code blocks and the collapsible information boxes solved this problem. The addition of the instructor's output to the test and grade results saved students from having to change all the code blocks so that they used the instructor's code and then running a test case in the "Test Your Program" form.

4 OTHER MODIFICATIONS THAT IMPROVED AUTOMATIC CODE ASSESSMENT

Several non-interface modifications involving problem presentation, an appeals process, and practice with Code Assessor were also made that have improved the student experience with CodeAssessor.

4.1 Problem Presentation

One interesting finding during Fall Semester 2012 is that students like to have a paper version of the problem in front of them. On the third and final exam of the semester, we gave each student a duplicate, paper copy of the problem to which

they could refer. This change was well-received by the students. Students also commented that they did not like the often verbose instructions that we provided in Code Assessor, since they then had to move back and forth too frequently between their code blocks and the instructions. We addressed these complaints in two ways on our first exam in Spring Semester 2013:

1. We moved the full-length description of the problem to the paper version.
2. We moved elements of the description that directly applied to a particular code block to the instructions for that code block.

The combination of these two approaches allowed us to considerably decrease the size of the initial instructions in Code Assessor and also to pair directly relevant instructions with each code block.

4.2 Appeals Process

One of the students' biggest complaints with the initial version of Code Assessor is that it was all or nothing. If the student's code almost worked, but missed one or two small elements, they still received a 0 for that part of the problem. We therefore initiated an appeals process for Spring Semester 2013, whereby students could ask the instructors to examine particular code blocks that students felt were nearly correct. This addition was very successful on the first exam. There were 42 appeals out of 168 test-takers with a median increase in score of 5, from 14/25 to 19/25. It was very easy for the instructors to quickly examine the code blocks because: a) they only had to look at code that was questionable, as opposed to a student's entire program, and b) they code was type-written, rather than in the student's handwriting, so the instructor could scan the code much more easily and quickly. Students were very appreciative of this appeals process, with most of them writing quick thank you notes expressing their gratitude.

4.3 Practice

Although we gave students a chance to use Code Assessor before the first exam in Fall Semester 2012, students displayed some difficulty getting "around" Code Assessor during the exam. Hence we had the students practice on Code Assessor at least twice during lab before the first exam was given in Spring Semester 2012.

5 CONCLUSIONS

The table in Figure 3 documents the steady increase in student outcomes using CodeAssessor as Fall Semester 2012 progressed. Some of the improvement is certainly due to the weaker students dropping the course as the semester progressed. However, the considerable jump in the median grade between the second and third exams suggests that at least some of the improvement was due to the improvements to Code Assessor described in this paper. Even more convincing is the comparison between the statistics for the first exams in Fall Semester 2012

and Spring Semester 2013. The median went from an 8 to a 14 before the scores were adjusted by the appeals process, and from an 8 to a 19 when the scores were adjusted by the appeals process.

Exam	Median	Average	Test Takers
F12 Exam 1	8	11.64	270
F12 Exam 2	11	13.04	240
F12 Exam 3	19	17.06	213
S13 Exam 1 (without appeals)	14	14.62	168
S13 Exam 2 (with appeals)	19	16.31	168

Figure 3: Results of exams taken using Code Assessor.

REFERENCES

[1] B. Vander Zanden, D. Anderson, C. Taylor, W. Davis, and M. W. Berry.Code Assessor: An Interactive, Web-Based Tool For Introductory Programming. In The Journal of Computing Sciences in Colleges, Dec. 2012, 28(2), pp. 73-80.

[2] Kumar, A. N. Generation of problems, answers, grade, and feedback—case study of a fully automated tutor. *Journal on Educational Resources in Computing 5*, (3) (Sept. 2005).

[3] Kumar, A. N. Problets homepage. http://www.problets.org/, 2012.

[4] Orsega, M., Vander Zanden, B. T., and Skinner, C. H. Two experiments using learning rate to evaluate an experimenter developed tool for splay trees. In SIGCSE'11 ACM technical symposium on Computer Science Education (Dallas, TX, Mar 2011), pp. 225–234.

[5] Orsega, M., Vander Zanden, B. T., and Skinner, C. H. Experiments with algorithm visualization tool development. In SIGCSE'11 ACM technical symposium on Computer Science Education (Raleigh, NC, Feb 2012), pp. 559–564.

[6] Savitch, W. *MyProgrammingLab with Pearson eText – Access Card – for Problem Solving with C++*, 8th ed. Addison-Wesley Publishing Company, USA, 2011.

[7] Stepp, M., and Miller, J. Practice-it. http://webster.cs.washington.edu:8080/practiceit/, 2011.