IMPACT OF AUTO-GRADING ON AN INTRODUCTORY COMPUTING COURSE

Mark Sherman, Sarita Bassil, Derrell Lipman, Nat Tuck, Fred Martin
Department of Computer Science
University of Massachusetts Lowell
Lowell, MA 01852
978-934-{1964, 3911, 1964*}
{msherman, sbassil, dlipman, ntuck, fredm}@cs.uml.edu

ABSTRACT

This project presents and assesses the impact of a new teaching tool that was deployed in an introductory Computer Science course. The tool is a web site that accepts student submissions for assignments, automatically tests them for correctness, and presents immediate feedback. Students can use that feedback to improve their work, and submit again. We compared student performance on assignments against previous semesters, which used the same assignments, but with no automated feedback system. We observed that students, when using the feedback system, make more submissions per assignment, indicating that students were leveraging feedback to improve their programs.

INTRODUCTION

We introduced a new system for auto-grading student programming assignments in an introductory computing course, and assessed the impact of that system on teaching. This system, named "Bottlenose," simplified the logistics of the grading process and helped students by providing near-immediate evaluation of their assignments.

The assignments in this course were programming exercises written in C. The course typically has 40 to 60 individual programming exercises, and 40 to 60 students in a course section. The development of this system was informed by researching systems developed at other universities, such as BOSS [5], CourseMarker [3], and Web-Cat [2]. Each of these systems was built to fit the nuances of a particular course or department, which made them difficult to customize for different pedagogical and technological environments. We created a system that is a general-purpose testing and evaluation platform for coursework that is devoid of language and testing preferences.

In the existing course, students are assigned programs approximately every lecture (multiple times per week). It is important to get feedback to the students as soon as possible, and this pace (coupled with the large enrollments of 50 to 60 students per section) creates a large workload for the instructor and the graders.

This is not only a logistical challenge, but also a pedagogical problem. Before this system, a student only saw feedback of an assignment after completing a number of subsequent assignments. This delay of feedback was detrimental to students, inhibiting their ability for informed iteration.

The system was designed to achieve these features:

1) Accept student code.
2) Run tests cases on student code.
3) Allow any testing methods, additional framework, or programs the instructor desires in tests.

4) Display test results to the student nearly immediately.
5) Store the results of test cases for instructor (or human grader).
6) Store students code for manual review by instructor (or human grader).
7) Provide efficient interface for grader to review tests, read code, and generate feedback.

This is a common situation for institutions teaching computer science [4]. The above features are aligned with the desirable features described by Ala-Mutka [1]. The system described in this paper, Bottlenose, represents a secure, lightweight, and easy-to-use solution to this feature space. Specifically, this system is an online automatic assessment tool, which runs pre-made tests on student submissions, but also allows for easy human assessment of the code and the test results.

Technical Description of System

This system was initially developed to support the teaching of a "flipped" course, where students watch video lectures online before class to prepare for classroom questions and discussion. It also includes online submission and grading of programming assignments, which are useful functions for traditional courses, as is examined in this paper.

The system was built using the Ruby on Rails [6] web application development framework. Student submissions are stored on the server file system, and other data, including grades, are stored in a database. The application runs on a linux server, providing two main benefits: The application utilizes many linux security mechanisms, and, the tests for student code can access the variety of linux-based applications and tools.

A simple process for online submission of assignments is provided. Students are emailed unique authentication links that bring them to their list of assignments, and identify the students to the application. Assignments are submitted by uploading the programming code directly in the web browser. Bottlenose supports both assignments requiring submission of a single source file, and assignments requiring multiple files, submitted as a compressed archive file. The automated grading process begins immediately when an assignment is submitted, giving students feedback within a few seconds. Students may attempt submissions multiple times.

In order to automatically grade student programs, submissions are compiled and run on the server. Allowing students to run arbitrary code on the server is clearly a potential security issue [4]. Bottlenose uses a sandbox mechanism to prevent student programs from causing trouble. Five major techniques are used to isolate student programs from the rest of the system:

1) **Separate system user** - Each student program is run under a separate, limited system user.
2) **Run in a sandbox** - Student programs can only access specific, white-listed parts of the file system.
3) **Disposable working directory** - Each program is executed in a temporary file system, which ceases to exist when the grading process is finished.
4) **Resource limits** - Limits on a variety of resources, including RAM, child processes, and created file size, are enforced on student programs.

5) **Watchdog timer** - A grading process is terminated if it lasts more than a set time limit.

This sandbox mechanism may be vulnerable to a clever student intentionally trying to defeat it. Thus far we have not yet identified any exploit. It does perform adequately at preventing the grading server from being disrupted by common student mistakes, like infinite memory allocation loops, without the need for exotic isolation systems, virtual machines, or additional servers.

Writing Tests

In preparation for student submissions of assignments to the system, the instructor writes a set of test scripts, which automatically evaluate specific aspects of students' programs. A test may evaluate, for example, that given specific input, the program generates proper output; it may check whether certain required functions are provided by the student's program; it can determine whether the program exited successfully or crashed; etc.

Tests can be written in any language. The only requirements for tests are that they conform to the Test Anything Protocol (TAP) [7]. With TAP, a test outputs an indication of the number of tests that will be run, e.g., 1..4 for four tests, followed by a line of output for each test, which indicates whether the test, e.g., test #2, succeeded (ok 2) or failed (not ok 2). The tests are otherwise free to provide input in any form to the program under test and to retrieve output via their standard output mechanism or via a file. Tests are also free to provide additional output, such as a description of the test to be run, or student's and expected output, when a program does not yield the correct results.

The feedback that students receive is entirely written by the instructor, and is generally displayed as a function of the success of failure of a particular test. To present feedback to the student, the instructor simply prints to screen in the test script, so the instructor may write feedback text based on any programmatic conditions.

Given the flexibility of this testing harness, it is possible to run some interesting tests on students' programs to let them know that they are correctly accomplishing their program's goal. The most common test we have written parses the output of the student's program. Parsing could be a direct comparison, a regular expression, or a custom program to process complex output. We have modified the student's source code and executables, replacing system calls with reference implementations that provide feedback of their use. We have isolated and unit-tested functions in the student code. We ran some student programs inside memory-checker programs, and parsed the output of the checker and returned it to the student. These examples are a small subset of all the testing mechanisms that are possible with Bottlenose.

The simplicity of this platform allowed for any test to be written that can be realized through another program, making the testing mechanism more flexible and robust than those used in other C-language studies, which largely depend on comparing student code and results against a reference implementation, such as Sterbini and Temperini [8] did, among others. With Bottlenose, the tests are written by the instructor to custom-fit the assignment, which can require a significant investment in time and effort by the instructor, but provides unsurpassed flexibility. Once written, tests and assignments can easily be re-used for subsequent course offerings.

METHODOLOGY

Analysis was performed on historic course data, with IRB-approved protocols to de-identify data and protect participants. The data included grades the students received on their submissions to assignments, as well as the submissions themselves, which were C source code files.

Two forms of analysis took place: (1) Aggregate analysis, tallying all submissions per assignment, per course offering, resulting in numbers of student submissions per assignment for each course section, and (2) fine-grain submission rate analysis, where the distribution of submission rates among students was visible.

The analysis used data from seven offerings of the same introductory computing course, spanning four semesters and three instructors. The data included participating students, defined as those who made homework submissions. Assignments that were not programming assignments were also removed from the data, such as in-class paper exercises, quizzes, and tests. All sections of the course used a common, core set of assignments. Assignments that deviated greatly from the common set, such as those with novel specifications or dependant on new concepts, were removed from the data.

RESULTS

Aggregate analysis showed a substantial increase in the number of submissions students made when using Bottlenose, compared to the no-feedback electronic submission system. Shown in Table 1, course sections M1-M4 were electronically submitted but manually graded, and sections A1-A3 used Bottlenose. The numbers of total submissions made, participating students, and assignments were divided to create the "Submissions per Student per Assignment" descriptor, which indicated the general rates of submissions per assignment in each section. A value of 1.0 would indicate that, on average, every student made one submission for every assignment. For any given assignment, there was a subset of students who did not submit an attempt for it, which lowered the average submission rates.

Course Section	# Submissions	# Students	Submits/Student/Assignment	Average	Difference
M1	2711	49	0.9706	0.9831	1.829x
M2	2672	47	0.9974		
M3	2425	50	0.9898		
M4	386	44	0.9747		
A1	3578	45	2.0387	1.7982	
A2	3119	47	1.7016		
A3	2561	36	1.6544		

Table 1: Submission rates among course sections

We also used a finer analysis technique, where the average submission rate per assignment was computed for each student. With this, we could see the distribution of submission rates within the course sections. In Figure 1, we can see the changes in that distribution between the sections that used Bottlenose (automatic) and those that did not (manual). The sections that used bottlenose show a slightly wider, right-tailed distribution, with the peak at a higher submission rate than the manually graded sections.

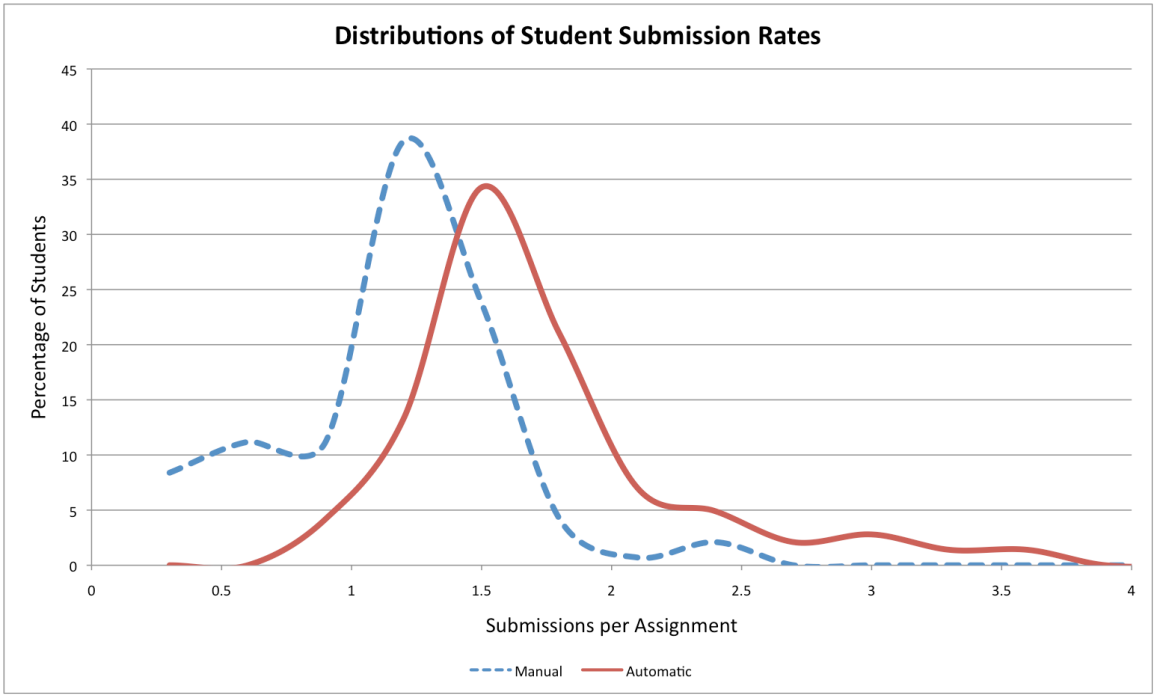

Figure 1: Distributions of how many students had similar submission rates, with and without the automatic Bottlenose feedback and submission system.

CONCLUSIONS

The standard deviation of the Submission/Student/Assignment rates of the manually graded sections, as seen in Table 1, was 0.013, indicating that, despite spanning multiple semesters and multiple instructors, submission rates before the introduction of Bottlenose were consistent. The course sections that used Bottlenose showed higher submission rates than those that did not. This increase was significant, showing that students, on average, made nearly twice as many attempts on an assignment when using the feedback-providing automatic assessment system.

Figure 1 shows the distribution of submission rates for sections with and without Bottlenose. Note that the courses that were manually graded showed a large portion of the students submitting, on average, just over once per assignment for every assignment. In those sections, almost none of the students had averages above two submissions per assignment. In the sections that used Bottlenose, a greater number of students submitted more than once, and many more submitted more than 1.5 times per assignment. Some students submitting more than twice per assignment, or more, with Bottlenose, which occurred rarely in sections without the feedback system. The lack of significantly higher

average submission rates indicates that students are not using it as their only compiler, or abusing the re-submission mechanism, which were common concern in the literature.

In total, this figure indicates a healthy change in mindset towards submitting, where students are using the system to help them make one or two additional iterations on their program.

FUTURE WORK

There is a wealth of data from this study remaining to be analyzed, including the quality of student iterations and products. In using this system, many tests were written by the instructors, which include the feedback that the students see. There may be relationships between the types of tests used, the quality and kind of feedback provided to the student, and student performance. This avenue of study is promising. There are also numerous case studies to be extracted from the data, which show individual students using the feedback from Bottlenose to inform their iteration, which will be presented in a future publication.

This system will be deployed in an upcoming semester in a "flipped classroom" graduate course, which may produce data in a different context, but still demonstrate connections between tests, feedback, and the student.

ACKNOWLEDGEMENTS

We offer special thanks to Prof. Anne Mulhern for reading and reviewing the paper for us. Thanks also go to James DeFilippo for his work on the auto-assessment prototype project.

REFERENCES

[1] Ala-Mutka, K., A survey of automated assessment approaches for programming assignments, *Computer Science Education,* 15, (2), 83-102, 2005.

[2] Edwards, S., Perez-Quinones, M., Web-CAT: automatically grading programming assignments, In *Proceedings of the 13th annual conference on Innovation and technology in computer science education (ITiCSE '08),* 2008.

[3] Higgins, C., Gray, G., Symeonidis, P., Tsintsifas, A., Automated assessment and experiences of teaching programming, *ACM Journal on Educational Resources in Computing,* 5, (3), 5, 5, 2005.

[4] Ihantola, P., Ahoniemi, T., Karavirta, V., Seppälä, O., Review of recent systems for automatic assessment of programming assignments, In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10)*, 86-93, 2010.

[5] Joy, M., Griffiths, N., Boyatt, R., The BOSS online submission and assessment system, *ACM Journal on Educational Resources in Computing,* 5, (3), 5, 2, 2005.

[6] Ruby on Rails, http://rubyonrails.org/ , retrieved November 15, 2012.

[7] Schwern, M., Lester, A., Documentation for the TAP format, 2003, http://search.cpan.org/~petdance/Test-Harness-2.64/lib/Test/Harness/TAP.pod , retrieved November 15, 2012.

[8] Sterbini, A., Temperini, M., Automatic correction of C programming exercises through unit-testing and aspect-programming, In *Proceedings of the 2nd International*

Conference on Educational Information Systems, Technologies, and Applications (EISTA '04), 6, 2004.