Quantitative Evaluation of Student Engagement in a Large-Scale Introduction to Programming Course using a Cloud-based Automatic Grading System

Narges Norouzi
Computer Science department
University of California at Santa Cruz
Santa Cruz, USA
nanorouz@ucsc.edu

Ryan Hausen
Computer Science department
University of California at Santa Cruz
Santa Cruz, USA
rhausen@ucsc.edu

***Abstract*—In this WIP Research to Practice paper, we explored the impact of integrating the university's learning management system and an automatic grading system in delivering a large-scale introduction to programming course in 2 consecutive quarters. Our initial approach utilizes an on-demand standalone automatic grading system and a separate assignment submission portal on Canvas. After evaluating our performance and specific student feedback, we integrated the assignment submission portal with the autograder system to provide a real-time objective assessment of assignments.**

The main improvement after enforcing assignment submission through the autograder (Stepik) was the noticeable improvement in the class average of assignment scores by 20.5% even though most of the test cases were hidden. Another interesting observation was the effect of our approach in decreasing the DFW rate to 12.5% from 46% and a considerable increase in the passing rate of female students, by 22%. We also noticed that in the second iteration of the course students who took the course as an elective were able to perform comparably and even better than students who took it as a requirement. It is also worth mentioning that using autograder helped students increase their code quality.

Keywords—Introduction to Programming, Java, Autograder, Grading System, Learning Management System, Computer Science

1. INTRODUCTION

In introductory courses in Computer Science (CS) accurate, meaningful, timely, and efficient feedback on programming assignments is an essential part of the learning experience [1], [2], [3]. The timely assessment of assignments is also important from the teacher's point of view since it can provide him/her with feedback that shows how learning goals are being met [1].

Another important factor in delivering successful introductory CS courses is using effective tools and techniques in designing large-scale courses. Based on a recent survey by the Computing Research Association (CRA), universities and colleges are facing a significant increase in enrollment in undergraduate CS courses [4]. The surge in CS enrollment needs to be addressed from different perspectives including use of scalable tools for assessment, designing the course to increase the retention rate of women and underrepresented minorities in CS, and making sure the passing rate for non-major students is comparable to that of major students.

There have been prior research studies on the design and features of autograder systems including studies on the architecture and security concerns when integrating Learning Management Systems (LMSs) with cloud-based autograders [5], [6], [7], [8], [9], [10]. Our approach in evaluating autograders is to investigate how to address the concerns about increasing enrollment in introduction to programming courses mentioned above, namely, passing rate of female students, performance of students who took the course as an elective(non-engineering students), and designing an effective and scalable assessment tool.

We offered two iterations of an intermediate programming in Java course during consecutive quarters. We investigated the impact of integrating a cloud-based automatic grading system using Stepik with the assignment submission portal on university's LMS. The integration improved the quality in delivering our large-scale programming course and more

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

effectively engaged students in the process of finding and working through errors before the assignment due date.

In the following section we describe the outline of our course. Section 3 discusses our methodology and set up of our autograder system implemented using Stepik. In section 4, we provide a quantitative evaluation of the major highlights of our study.

2. COURSE OUTLINE

This experiment was conducted during two quarters of teaching of the intermediate programming in Java course to 352 students with different levels of programming background. This course is a continuation of another introductory to programming course and therefore students are assumed to have a prior experience with coding in Java. Table I shows some statistics of both courses, including 3rd week enrollment, DFW rate (the percentage of D, F, and W grades), percentage of female students, and percentage of students who took the course as a requirement (engineering students) enrolled in each one of the courses.

TABLE I. ENROLLMENT INFORMATION OF TWO COURSES

	3rd week enrollment	DFW rate	% of Female students	% of non-engineering students
Original Course	112	46%	29.5%	44.6%
Revised Course	240	12.5%	23.3%	20%

Both iterations of the course consisted of 5 programming assignments designed around various topics ranging from basics of Java programming concepts to more advanced topics in Object-Oriented Programming (OOP) such as inheritance, polymorphism, interfaces, etc. Table II outlines weekly lecture topics and deliverables. For the purpose of this experiment, we wrote autograder modules for the first 4 assignments.

TABLE II. COURSE SCHEDULE AND CONTENT

Week	Lecture Topics	Deliverables
1	Review/Operators/Conditionals/Loops	
2	Arrays/Methods/Recursion	Assignment 1
3	Strings and characters	
4	Object-Oriented Programming	Assignment 2
5	Inheritance/Static & Final	
6	Abstraction/Interfaces	Assignment 3
7	Polymorphism	
8	Collections	Assignment 4
9	GUI	
10	ActionListener/Inner Classes	Assignment 5

3. METHODOLOGY

Our initial approach in designing an assessment tool for the assignments utilizes an on-demand standalone automatic grading system using Stepik and a separate assignment submission portal on university's LMS, Canvas [11]. Students were able to test their code as often they wanted without any penalty before the due date (unintegrated). After evaluating our performance and specific student feedback, we integrated our assignment submission portal with Stepik to provide a real-time objective assessment of assignments (integrated). Furthermore, this integration helped us in removing any ambiguity in assignment grading rubric.

I. Automatic Assessment (AA) of Assignments using Stepik

The Stepik autograding system provides an open and flexible approach to the implementation of an autograder for programming assignments. A rubric can be as simple as giving a series simple inputs/outputs and can be as sophisticated as writing unit-test style tests in Java. One of the aspects of the Stepik system that is very powerful is the ability to generate tests and validate student output in Python. This is powerful because you can introduce randomness to the inputs, but it also allows for the validation of student output in a more sophisticated manner. If, for example, you want to award credit for answers within a numerical tolerance or if there is more than one correct output for a given input.

From the student perspective Stepik also seems intuitive to work with. If there are compilation or runtime errors within the student's code those errors are displayed to student. If the student's code runs, but is still incorrect, then the given input, the student's output, and the expected output are displayed to the student. In addition, you can have tests that are 'hidden' to the student, which will just return a failure message and no detail of the program output or correct answer. There isn't enough space in this paper to accommodate the details of the variety of implementations, but we have created a brief tutorial demonstrating some of the high level features [11].

Some considerations to review before using Stepik for a class are your IT infrastructure and flexibility with respect to external LTS integration. Further, its important to consider student confidentiality and your course content IP. Stepik offers public and private courses. Public courses are free to

create and use, but the course information and content is open to the public, including who's taken the course. Private courses don't have this restriction, but also cost a fee. For our implementation we used private courses as student privacy was a concern.

4. RESULTS

Major highlights of our quantitative evaluation of both iterations of our large-scale intermediate programming in Java course consist of the following:

I. Increase in average of assignment grade as well as the final course grade

The main improvement after enforcing assignment submission through Stepik was the noticeable improvement in the class average of assignment scores by 20.5% even though most of the test cases were hidden from students. This suggests that we were able to engage students more in the practice of evaluating their logic and thinking about all possible test scenarios.

Fig.1. Illustrates a comparison between average of assignment scores and the final course grade for both versions of course. As shown, average assignment scores and the final course grade are higher after integration of autograder and LMS.

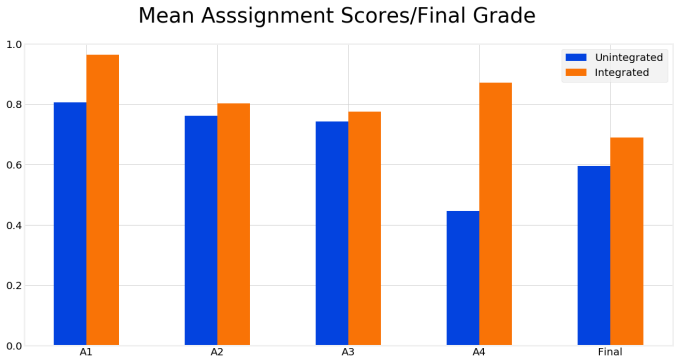

Fig. 1. *Comparison of average assignment scores and the final course grade for the both versions of the course*

Another interesting observation was that as students moved to more involved topics (basics of OOP, inheritance, and polymorphism), both the number of correct submissions on Stepik and the class average of the assignment dropped in the unintegrated of the course. However, in the integrated of the course the class average for different assignments remains in the same range for different topics, but we saw more assignment submissions on Stepik for more advanced topics. Specifically, in assignment 4 which was designed around polymorphism, interfaces, and inheritance the class average improved by 43%.

II. Increase in code quality

One reasonable expectation of not limiting the number of submissions allowed for grading is that the students may try develop and debug their code using the autograder. A closer look at the student submissions indicates that this seems to not be the case when the students are required to use the autograder. Students who were required to submit their code through the autograder were observed to have noticeably fewer compilation errors on average than the students who used the autograder voluntarily. They also tended to have fewer runtime errors. Fig. 2 shows the mean number of submissions per student that were rejected for either a compilation or a runtime error for each assignment in the course.

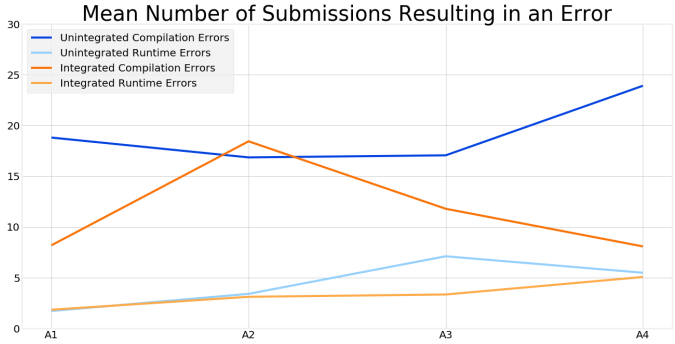

Fig. 2. *Comparison of average number of rejected submissions per student because of compilation or runtime error*

III. Engagement of female students

Another interesting observation was that after integrating the autograder system with our LMS, we noticed that the average score on assignments for our female students increased by a factor of 6.25% compared to their male peers. Furthermore, the passing rate of our female students increased from 60% to 82% in the second iteration of the course. Fig. 3 shows the comparison between average assignment scores for our male and female students in both courses.

On a separate note, we also noticed that before integrating submission of assignments through Stepik, female students tended to test their code more.

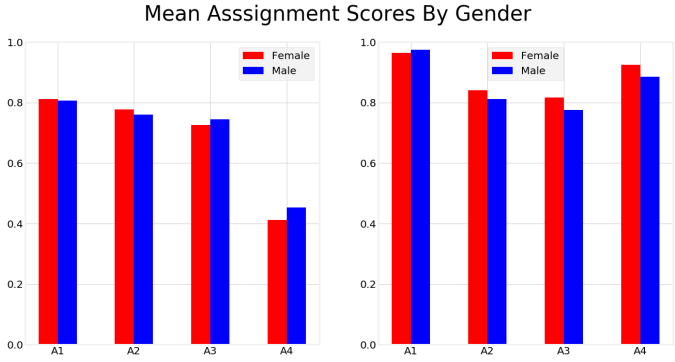

Fig. 3. *Comparison of assignment averages for male and female students in the unintegrated course (left) and the integrated course (right)*

IV. Engagement of non-engineering students

One of the main factors that makes a successful programming course is its effectiveness to convey CS materials to non-major students. For the sake of this study, we define a student to be an 'engineering student' if the course is required within the student's major. An example of this would be an Electrical Engineering student. Fig. 4 compares the performance of engineering and non-engineering students in both courses. As shown below, non-engineering students outperformed major students in all of the 4 assignments when integrating submissions with Stepik.

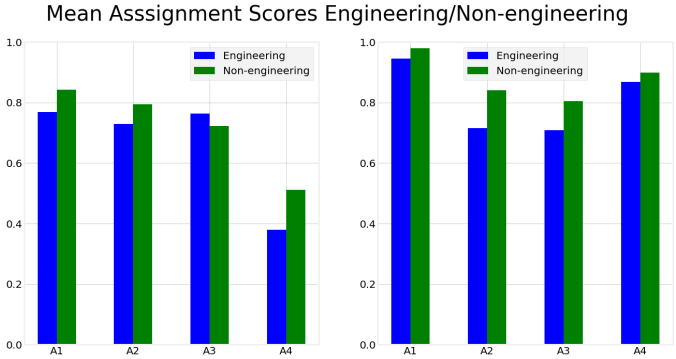

Fig. 4. *Comparison of assignment averages for engineering and non-engineering students in the original course (left) and the revised course (right)*

V. Correlation between the average of assignment grades and the number of Stepik submissions

The main motivation of this study started from noticing a moderate positive correlation (R = 0.39) between the number of times students were using Stepik to test their code and the average of their assignment grades. Fig. 5 shows the scatter plot of the average assignment grades and number of Stepik submissions in the unintegrated version of the course, color-coded by reported gender of students. As shown here, female students tend to evaluate their code more. We were not able to see the same correlation in integrated course, mainly because as students got familiar with testing strategies, they did not need to have as many submissions to get full credit on the assignment.

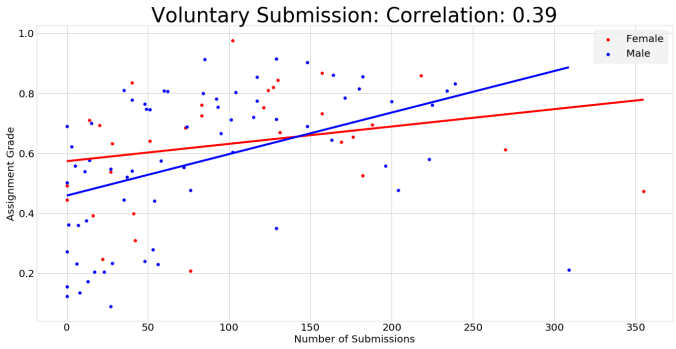

Fig. 5. *Correlation between average assignment grade and the number of submissions on Stepik for both versions of the course, color-coded by gender*

5. CONCLUSION

In this paper we analyzed the effect of integrating an autograder system with university's LMS in our introduction to programming course during 2 consecutive quarters.

Based on quantitatively evaluating both versions of the course, we reported the effectiveness of the integrated submission of assignments through autograder in decreasing DFW rate (from 46% to 12.5%), increasing passing rate of female students (from 60% to 82%), and engaging non-engineering students to have comparable and better assignment scores than engineering students.

Furthermore, average assignment scores for students in the integrated version of the course increased by 20.5%. The same students also had a showed an increase in their final course grade by 9.5%. Last but not the least, using autograder helped students increase quality of their code and as a result we observed less rejected submissions due to compilation or runtime errors.

6. FUTURE WORK

The future direction of this research will focus on evaluating the effect of the different features of autograders in students' success in programming courses. Important features to be explored include resubmission policy, approaches in writing a test suite, and the structure of the feedback.

REFERENCES

[1] Ihantola, Petri, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppala. "Review of recent systems for automatic assessment of programming assignments." In Proceedings of the 10th Koli calling international conference on computing education research, pp. 86-93. ACM, 2010.

[2] Nordquist, Pete. "Providing accurate and timely feedback by automatically grading student programming labs." Journal of Computing Sciences in Colleges 23, no. 2 (2007): 16-23.

[3] DeNero, John, and Stephen Martinis. "Teaching composition quality at scale: human judgment in the age of autograders." In Proceedings of the 45th ACM technical symposium on Computer science education, pp. 421-426. ACM, 2014.

[4] "CRA Releases Report on Surge in Computer Science Enrollments" *CRA*. CRA, 24 February 2017. Web. 1 May 2018.

[5] Danutama, Karol, and Inggriani Liem. "Scalable autograder and LMS integration." Procedia Technology 11 (2013): 388-395.3.

[6] Helmick, Michael T. "Interface-based programming assignments and automatic grading of java programs." In ACM SIGCSE Bulletin, vol. 39, no. 3, pp. 63-67. ACM, 2007.0.

[7] FERNANDO, Jordan, and MM LIEM. "Components and Architectural Design of an Autograder System Family." Olympiads in Informatics 8 (2014).

[8] Kunchala, Ashrita, and Maneesh Gunnala Ranga Rao. "Java Auto Grader." (2016).

[9] Davuluri, Prathibha, and Pranavi Reddy Madadi. "Moodle Java Autograder." (2016).

[10] Zimmerman, Daniel M., Joseph R. Kiniry, and Fintan Fairmichael. "Toward instant gradeification." In Software Engineering Education and Training (CSEE&T), 2011 24th IEEE-CS Conference on, pp. 406-410. IEEE, 2011.

[11] Norouzi, Narges, Hausen, Ryan. “Stepik: FIE 2018 Grader Tutorial”. Web. 1 June 2018. https://stepik.org/course/13640/syllabus