Building Autograders for Open-Ended Assignments in a Computer Science MOOC

Arunima Suri, University of Illinois Urbana - Champaign
July 26th, 2024

Faculty Mentors:
Dr. Susan Rodger, Duke University
Dr. Kristin Stephens-Martinez, Duke University
Yesenia Velasco, Lecturer, Duke University

Undergraduate Team:
Nikita Agarwal, University of Wisconsin - Madison
Kevin Alvarenga, Duke University

1 Motivation

Open-ended assignments help students to be creative and enhance their critical thinking skills. These assignments encourage students to apply their existing knowledge in innovative ways. However, it is time consuming for instructors to manually grade these open-ended assignments. It is beneficial, especially in large scale courses, to have autograders that can provide immediate feedback and allow students to reflect on their project in a timely manner. This efficiency not only enhances the student learning but also alleviates the workload on instructors, enabling them to focus more on creating the lecture content. While there are many benefits to using autograders, most of these autograders are meant for assignments with a single goal/end state. This can work well in higher level CS courses where there is one main objective, or one way to solve a specific problem. However, open-ended assignments, which allow for more creativity and can lead to more diversity in computing, are common in introductory Computer Science courses. There is a gap in research on autograders for these types of open-ended assignments.

2 Background and Related Work

2.1 The Use of Open-Ended Assignments

Significant research has been done on using open-ended assignments in introductory Computer Science classes to see how they can impact student motivation, self-efficacy, engagement, grades, and more. When referring to open ended assignments, different authors use slightly different definitions. A general consensus is that an open-ended assignment allows students to make their own choices and decisions about aspects of the project, which results in each student having their own solution to a general problem [9]. For example, an open-ended

assignment may ask a student to create a game given general requirements such as: the game must have one to five user inputs, each input may only take in the numbers one or two, once the game is over the program should print if the game is a win or a loss, the program should ask the user if they want to play again, and the program should display a summary of the game rounds once the user chooses to not play again. From these general requirements, students could create games such as guess the number, guess if the number is odd or even, solve a maze, implement wandering letters, and more!

The use of open-ended assignments is motivated by many different learning theories, including the idea that students build knowledge by looking at prototype models, the constructivist learning theory, and student metacognition [12]. Open-ended assignments have shown to have a positive impact on student motivation, confidence level, and satisfaction [9]. They also have had no impact on student grades, self-efficacy, or attention span. Thus, the research shows that open-ended assignments, at the very least, do not have potential negative effects. [9].

Open-ended assignments can be beneficial, but only if implemented correctly. There is still the potential issue of students having a lack of interest, which can affect their learning. In 2018, researchers were curious if letting students choose their assignment would aid with learning. After examining an introductory CS class, researchers noticed that many students did not enjoy the class because the assignments did not align with their interests. To address this, the researchers designed 5 open-ended assignments for students to choose from and were then able to notice that student enjoyment and strength on core concepts increased [1]. However, it was also mentioned that the grading process for these assignments required a significant amount of work and was tedious for instructors. The researchers noticed that incorporating some form of choice was beneficial for the students in their learning, while acknowledging that there is still room for improvement in the actual grading process. Therefore, the use of choice in these courses has the potential to provide benefits to the student, but there is still no path to designing autograding capable of evaluating these assignments.

2.2 The Efficacy of Autograders

Autograders have been helpful in courses with a large roster of students as they reduce the time needed for grading, give clear feedback, and can evaluate complex equations much quicker. Also, autograders can be used to help guide students to improved solutions with less errors. In terms of autograding open-ended assignments, research shows that generating hints for these assignments can be difficult due to variability in programs. However, there have been cases in which instructors were able to use student data to generate hints/feedback for the students, which helped maintain the core of open-ended assignments while ensuring that students still received the help they needed [8]. However, there has been less research conducted on the student side of things. Student perceptions on autograders can affect their performance, experience, and approach to completing assignments with some students specifically stating that they would have had a better experience with an autograder if they could directly cater their

programs for it [5]. However, this negates the benefits of using open-ended assignments as the main purpose is to promote creativity and new approaches.

A different approach to autograding was taken in 2022, where researchers developed an algorithm that could grade code based on different aspects, such as lines of code, spacing, variable names, string count, etc. [11]. This helped with course assignments that had visual and graphical output, which typically would be difficult to grade using an autograder. The algorithm focused on classifying the elegance of a student's code, which, while important, is not a sufficient criteria to use for grading open-ended assignments. This algorithm did not take into account the actual output of a student's code, which is the main concern when it comes to open-ended assignments.

Given the previous research out there, it is clear that there lacks an integration of autograders into open-ended assignments for introductory computer science classes. And with these classes having larger and larger student enrollment, assignments simply cannot be graded by hand. Therefore, it is critical that autograders be introduced for open-ended assignments as open-ended assignments are an important way to ensure that students are able to be creative and stay motivated as they complete assignments. In other words, the main problem lies in a lack of existence of autograders that can handle student creativity, so our work aims to fill this gap through building and documenting autograders for open-ended assignments [10].

3 Implementation

The main focus of my work was using python, and its testing library, pytest, to build autograders for open-ended assignments for an introductory college-level computer science course in Coursera. Coursera is an online course system that allows users to go at their own pace. Also, it allows for instructors to create their own grading systems for particular assignments. My team and I created multiple autograders for assignments in the course and implemented them directly into Coursera. In using open-ended assignments for the course, students would be able to use their creativity and stay engaged with the programs they write, and by using autograders, the assignments would be automatically graded so that instructors do not need to take extra time to grade hundreds of student solutions. There is currently little research that focuses on designing an autograder that can effectively grade and provide feedback for more creative open-ended assignments.

3.1 Autograder Creation

Each lab and assignment has a specific set of requirements that students are given as a guideline for how their work will be evaluated. These requirements are split into two categories: satisfactory and exceeds expectations, with the satisfactory requirements making up 80% of the points and the exceeds expectations covering the last 20%.

The autograder build was split up to reflect this breakdown, with three different sets of test cases: pre-tests, required tests, and exceptional tests. The pre-tests consisted of checking the student code for baseline issues, such as compile time errors, runtime errors, and too few or too

many calls to user input. The required and exceptional tests were written to essentially reflect the set of requirements laid out in the lab checklist for satisfactory and exceeds expectations, respectively. The test cases were written using mainly the pytest library, with the help of some other libraries such as regular expressions. The order that the test cases were written in reflected the order that the student code would be tested against those cases, with the student receiving the message of the first test case that they failed.

In order to validate the test cases, sample student solutions were crafted to reflect typical logical or syntax errors students could make when writing their code. The student solutions were run against the test cases to ensure that students would be given the correct message for the first test case that they failed so that they could improve on their solutions in a step by step manner.

Once the autograder was validated to work properly in admin mode, it was then tested in a student view to ensure that the Coursera platform was connecting with the autograder as intended. In this process, the starter file for the student solution was also generated so that students would have a foundation to build their code off of. This especially became important when working with functions as it was critical to ensure students had the correct function headers in their code.

3.2 Overview of Specific Autograders

Decision Advisor with Randomness

This autograder was created in collaboration with the other undergraduate researchers on the team. The lab tests a student's ability to work with boolean expressions, conditional statements, and the random library. This autograder followed the typical build process, with the main challenge being how to account for the randomness of the student code.

Art Generator with While and Indexing Part 2

This lab tests a student's ability to properly implement a while loop and its stopping condition as well as verifying user input and correctly indexing into a string. The student's code has to generate an art design based on user input that provides a string of alternating characters and numbers to produce a multi-line pattern.

Debugging

This autograder was created in collaboration with the other undergraduate researchers on the team. In this lab, students are given incorrect starter code for a temperature converter and asked to debug it. The code includes logical errors as well as syntax errors that the student needs to identify and fix. Even though the lab came with a partial solution, the autograder build followed the regular process.

Accumulator Pattern

This lab asks students to accumulate user inputs and produce three separate statistics regarding the data: the sum, the minimum, and a third statistic of their choosing. In doing so, the students

are tested in their ability to properly implement while loops, conditionals, and the accumulator pattern. The challenge for this autograder was in identifying that a student correctly included a third statistic without knowing what the statistic would be, and this was tested using regular expressions to extract all the numbers from the printed output and removing the sum/minimum to ensure a third number existed. Moreover by giving the student code a variety of inputs, the tests also had to ensure that the third statistic actually changed based on the input and did not just consist of a hard coded value.

Quiz Generator

This is the first lab of course 2 of the Coursera course, which means that it includes functions. This changed the entire autograder approach, not in terms of the general structure, but in terms of how the individual test cases were written because pytest has a different approach to testing functions. With this autograder came learning how to implement parameterization as well as how to capture not only the return value of a function but also any printed output that was produced while the function was run. The lab itself has three functions the student needed to implement to resemble the creation of a quiz that recorded the user's score.

Debugging Functions

This lab was similar to the debugging lab from course 1, but it involved functions. The lab also has three different starter files for the student, which ultimately was decided to be split into three separate autograders, one for each file. Once the decision was made to split the lab up into different autograders, the build process itself was very straightforward. The mini labs are debugging functions that collect vowels, print n-grams (sets of characters), and that identify the biggest of three numbers.

Temperature Visualizer

This was actually an assignment, not a lab, which means it is a larger task that has a more cumulative approach to see if students can apply multiple weeks of learning into one assignment. The students are asked to essentially create a set of functions that resemble the process of making a temperature blanket, with students reading in data from a csv file. The additional layer of difficulty came in as students are also required to create a set of test functions to test that their own code worked. This meant the autograder not only had to test the students' functions but also the students' test functions to make sure both files of functions met the requirements outlined in the lab. This process required learning a lot about the pytest mock functionality and how to apply that for mocking csv open and mocking calls to user input.

4 Proposed Research Question

Because the Coursera course is still in its creation phase, the focus of the work was not on data collection and data analysis. That being said, once the course is launched and data can be

collected, there are many opportunities for discovery on how open-ended assignments and the use of autograders impact student performance, experience, and more. One such area of interest for me is using the course to identify the differences between satisficers and maximisers in terms of what attributes can be linked to student performance in open-ended CS MOOCs?

4.1 Background for Potential Research Question

The post pandemic world has seen significant changes in the way that both students and instructors approach learning. Online and blended learning styles have become significantly more popular, especially in Computer Science, due to their ease of use and ability to accommodate significantly more users. In addition, the field of computer science, itself, has greatly grown in popularity with the introduction of AI, the increased use of programming in various other academic fields, and more generally the rise of reliance on technology. And with this increase in interest, there is an increase in demand for introductory Computer Science courses. Massive Open Online Courses (MOOCs) can easily fill this demand because they are readily available, flexible, and have a low barrier to entry.

With the ease that comes with learning through MOOCs, also comes a multitude of reasons why a specific individual may choose to enroll in an online course. These can range anywhere from a general interest in learning and receiving formal recognition for their knowledge to making social connections and preparing for future college courses or job prospects [2].

However just because an individual chooses to enroll in a MOOC course does not necessarily mean that they will choose to complete the course to the best of their ability, or even more simply, complete the course. This phenomenon lies as a consequence of there being less stakes in completing an online course, especially for courses that individuals do not pay for. That being said, many individuals do still choose to put in a significant amount of effort and complete online courses to the best of their ability. Essentially, enrollment in MOOCs can be categorized into those who choose to put in their full effort (maximisers), those who put in the bare minimum to pass the course (satisficers), and those who do not complete the course. In the case of this study, we will be looking specifically at the maximisers and the satisficers and what different attributes separate the two categories when it comes to an Open-Ended CS MOOC.

4.2 Motivation for Potential Research Question

Differentiating between the maximisers and satisficers for a CS MOOC will allow educators to better understand the individuals who take their courses, which in turn can inform how they choose to design their lesson plans. Educators are able to choose the criteria students need to meet in order to pass a specific course, so in knowing how different groups of individuals engaged with their lessons, they can cater the assignment requirements to specifically target certain groups. In turn, educators would be able to try their hardest to ensure that students come out of the course having learned certain skills.

Moreover by conducting this study in an Open-Ended MOOC, it will allow educators to understand if using Open-Ended versus Closed-Ended assignments has had any impacts on the typical trends that seem to be present in CS MOOCs in terms of how motivation, gender, race, among many other factors, impact completion rates. This could give insight into how students may engage differently depending on the types of assignments they are given.

4.3 Related Work

Previous research has been conducted for Computer Science MOOCs that look at course attributes and how they correlate to course engagement and completion rates. One major aspect that has shown major links to student success in online courses is active participation in course forums. Active participation refers to posting original comments, commenting on others' posts, or viewing the forum posts, and it meant an individual had a higher chance of staying engaged longer in the course [3]. Broken down even further, students who posted on the forum persisted longer in courses than students who did not post, and students who visited the forum persisted longer than students who chose to not view the forum page at all [2]. Gender has also shown to play an interesting role in CS MOOC persistence, with men being more likely to persist longer in courses than women [4].

Additionally an individual's motivation behind taking a CS online course has been linked to course completion rates. When broken down into clusters (opportunity motivated, success motivated, interest motivated, and over-motivated), opportunity motivated individuals had higher completion rates than success motivated individuals, and the rest of the categories showed no significant statistical differences [7].

4.4 Methods

Course Description

The course that is used for this study is a python-based CS-1 level course hosted on the Coursera platform. It is meant for users with little to no programming experience, making it accessible to a wide range of audiences. The course, itself, is broken into 4 different series, which sequentially get more advanced as the user begins to build on their learned skills. The students are assessed on their learning using weekly multiple-choice quizzes, labs, and assignments. The labs and assignments are open-ended, allowing for student creativity, and are also autograded, providing users with instant feedback messages for their programs. In order to pass the labs and assignments, users must score at least 80% of the total points, with the score more heavily weighted on meeting the fundamental learning objectives of each task and less weighted on the additional elements that allow for more complexity.

Data Collection

The data for this study includes (1) survey data collected prior to the beginning of each series in the course, (2) scores on labs and assignments (as a percentage), (3) number of submissions for labs and assignments, and (4) percentage of completion for the course.

The Pre-Course Survey: Participants of the survey are asked basic demographic questions, their prior programming experience, their motivations for taking the course, and their academic background. Completion of the survey is optional and does not affect a user's experience during any part of the course. For programming experience, users are given a scale of 1 to 5 from ranging from no experience to having two plus years of experience. Motivation is gauged using the Online Learning Enrollment Intentions Scale (Appendix A), in which users are asked to select the top two reasons that applied to them. For academic background, users are given a set of options (Appendix B) and asked to select the option that applies best for them.

Lab and Assignments Scores: Scores for each lab and assignment are a percentage based on the number of autograder test cases the participant is able to pass. Scores for labs/assignments that lack a variety of data (i.e. essentially all students score near a 100%) should be excluded from the data set.

Number of Submissions: Number of submissions for each lab/assignment is the count of the number of times the user hit the submit button and received feedback from the autograder.

Percent Completion: Participants' percent completion for each series in the course is the number of tasks they completed out of the total number assigned. Tasks include quizzes, labs, and assignments.

Data Analysis

The first step to data analysis for this study would be to determine the threshold that separates a student from being a satisficer versus a maximiser. By using statistical mapping to get a spread of the average lab and assignment scores, distributions can be made for each of the four course series. In viewing the data spread, a line can essentially be drawn for each series as to where the threshold lies, in which a student goes from being a satisficer to a maximiser. The reason to split this data differently for each series is that some series may have more individuals who were able to complete all the coursework than others. All the individuals who did not complete all the assignments for a series are not included in the distribution of the average scores because their average would be disproportionately lower than it should be.

Once the line is drawn between the satisficers and the maximisers, the attributes identified using the pre-course survey can be compared. This will give insight into what academic backgrounds, motivation groups, prior-experience levels, etc were more likely to be in one category or the other.

5 Conclusion

Autograding open-ended assignments for introductory Computer Science courses opens up a whole new era of advancement for Computer Science education. Not only does it ensure that students are allowed the freedom to be creative and innovative in their programming solutions, but it also ensures that educators are not bogged down in grading assignments. This will allow educators more time to develop curricula, work on research, or focus on any other aspect of their jobs. That being said because this is a new area of research, the effects on student learning are yet to be seen. As data is collected and analyzed, it will shed light on the impacts on student motivation, retention, completion, and more. While streamlining educators' work load is important, making sure that student learning is not negatively impacted is even more critical.

6 Appendix

Appendix A

Online Learning Enrollment Intentions (OLEI) Scale [6]

Why did you enroll in this course? (Pick two that apply)

- General interest in the topic
- Relevant to job
- Relevant to school or degree program
- Relevant to academic research
- For personal growth and enrichment
- For career change
- For fun and challenge
- To meet new people
- To experience an online course
- To earn a certificate/statement of accomplishment
- Course offered by prestigious university/professor
- To take with colleagues/friends
- To improve my English skills

Appendix B

Which area best describes your academic background? (Pick one)

- Computer Science
- Engineering
- Physical Sciences
- Life Sciences
- Social Sciences
- Business
- Math
- Humanities

- Arts
- Other

7 References

[1] Sohail Alhazmi, Margaret Hamilton, and Charles Thevathayan. CS for All: Catering to Diversity of Master's Students through Assignment Choices. ACM Technical Symposium on Computer Science Education, 2018.

[2] R. Wes Crues, Nigel Bosch, Carolyn J. Anderson, Michelle Perry, Suma Bhat, and Najmuddin Shaik. Who they are and what they want: Understanding the reasons for MOOC enrollment. International Conference on Educational Data Mining, 2018.

[3] R. Wes Crues, Nigel Bosch, Michelle Perry, Lawrence Angrave, Najmuddin Shaik, and Suma Bhat. Refocusing the lens on engagement in MOOCs. ACM Conference on Learning at Scale, 2018.

[4] R. Wes Crues, Genevieve M. Henricks, Michelle Perry, Suma Bhat, Carolyn J. Anderson, Najmuddin Shaik, and Lawrence Angrave. How do Gender, Learning Goals, and Forum Participation Predict Persistence in a Computer Science MOOC? ACM Transactions on Computing Education, 2018.

[5] Silas Hsu, Tiffany Wenting Li, Zhilin Zhang, Max Fowler, Craig Zilles, and Karrie Karahalios. Attitudes Surrounding an Imperfect AI Autograder. CHI Conference, 2021.

[6] René F. Kizilcec and Emily Schneider. Motivation as a Lens to Understand Online Learners: Toward Data-Driven Design with the OLEI Scale. ACM Transactions on Computer-Human Interaction, 2015.

[7] Piret Luik and Marina Lepp. Are Highly Motivated Learners More Likely to Complete a Computer Programming MOOC? International Review of Research in Open and Distributed Learning, 2021

[8] Thomas W. Price, Yihuan Dong, and Tiffany Barnes. Generating Data-Driven Hints for Open-Ended Programming. International Educational Data Mining Society, 2016.

[9] Sadia Sharmin, Daniel Zingaro, and Clare Brett. Weekly Open-Ended Exercises and Student Motivation in CS1. Koli Calling, 2020.

[10] Sadia Sharmin, Daniel Zingaro, Lisa Zhang, and Clare Brett. Impact of Open-Ended Assignments on Student Self-Efficacy in CS1. ACM Conference on Global Computing Education, 2019.

[11] Sirazum Munira Tisha, Rufino A. Oregon, Gerald Baumgartner, Fernando Alegre, and Juana Moreno. An Automatic Grading System for a High School-Level Computational Thinking Course. International Workshop on Software Engineering Education for the Next Generation, 2022.

[12] Tammy Vandegrift. Encouraging Creativity In Introductory Computer Science Programming Assignments. 2007 Annual Conference Exposition, 2007.