DO STUDENTS NEED DETAILED FEEDBACK ON PROGRAMMING EXERCISES AND CAN AUTOMATED ASSESSMENT SYSTEMS PROVIDE IT? *

Angelo Kyrilov & David C. Noelle
Electrical Engineering and Computer Science
University of California, Merced
5200 North Lake Road, Merced, CA 95343, USA
{akyrilov, dnoelle}@ucmerced.edu

ABSTRACT

This paper examines the degree to which binary instant feedback on computer programming exercises, provided by an Automated Assessment (AA) system, benefits students. It also offers an approach to providing improved feedback. Student behavior in an undergraduate computer science class was studied. Students were assigned exercises requiring the generation of programs that met given specifications. We employed an AA system that evaluated the correctness of student code by executing it on a set of test cases. Students promptly received binary ("Correct"/"Incorrect") feedback, and they could repeatedly resubmit solutions in response. We found that more than half of the students failed to achieve correct solutions within a reasonable time. A small group of students were also found to have plagiarized solutions. This result led us to investigate ways in which AA systems for programming exercises might provide more rich and detailed feedback. We propose the development of clustering algorithms that group solutions based on how *similarly incorrect* they are. For the exercises we considered, there were, on average, 64 incorrect submissions, but there were only 8-10 distinct logical errors. This means that, if all incorrect submissions were automatically grouped into 8-10 clusters, a human instructor would only have to produce detailed feedback once for each cluster. That feedback could then be automatically delivered in response to each submission that fell within that cluster. We provide evidence that such an

---

approach would result in substantial labor savings, while providing instant detailed feedback to students.

1. INTRODUCTION

In order to address the problems associated with increasing class sizes in undergraduate computer science courses, many instructors have turned to Automated Assessment (AA) systems for computer programming exercises. There is extensive support for AA systems in computer science education [1, 5]. In addition to reducing the assessment workload for instructors, AA systems offer a level of objective consistency in program evaluation that is difficult for human instructors to achieve. AA systems may be accessible online, allowing students to utilize them from virtually any location and at any time. Some researchers, however, have warned of potentially undesirable effects of using AA systems for programming exercises [2]. Chief among the concerns is that the low quality of the feedback provided to students by such systems, often indicating only if a submitted program is correct or incorrect, may actually hinder student learning. Most educators acknowledge the shortcomings of AA systems, but they continue to use them, supposedly driven by the belief that the advantages outweigh the disadvantages.

We have been using an AA system for programming exercises in undergraduate courses. This AA system was developed in-house, and it uses a test-based approach to determine the correctness of student submissions. When designing and implementing a programming exercise, the instructor provides a number of test cases, providing program input and specifying correct program output for each case. A student submission that passes all test cases produces a "Correct" message to the student, while submissions that fail one or more tests result in an "Incorrect" message. In order to guard against non-terminating submissions, if a student's code begins to use too many resources during any execution, the program is halted and a notification of such is sent to the student. Finally, if a submission fails to compile, any compiler messages are forwarded to the student.

The data for this study was collected from a semester-long undergraduate course on programming in C++. The first research question we addressed was related to test-based AA systems for programming exercises, such as ours. Specifically, *"To what extent do such systems help students discover and remove errors from their code?"*

We found that almost half of the students who initially submitted an incorrect solution to an exercise corrected their work quickly, suggesting that the feedback they received was useful. The other half of the students, however, needed large amounts of time to arrive at a correct solution, which typically left them insufficient time to attempt every assigned exercise. There was a small number of students who gave up completely, abandoning attempts at further exercises in a given set, even though there was available time. Other students resorted to academically dishonest practices.

If the successful completion of programming exercises is seen as important for learning, these results suggest that the "Correct/Incorrect" feedback that we provided was inadequate to support student learning for more than half of the students. We conjecture that these students would substantially benefit from the prompt delivery of more detailed feedback. Along with many computer science educators, we suspect that the optimal way to generate high-quality feedback for incorrect submissions is to have an expert human

instructor examine the code and generate pedagogically appropriate comments. However, given the typically large class sizes in undergraduate computer science, it is impractical for a teaching assistant to debug every incorrect submission received by the system and rapidly provide detailed feedback to the student.

This lack of labor resources led us to seek ways to more productively leverage educator expertise. In the second part of our study, we examined the specific kinds of programming errors that students made, determining if it is possible to group together incorrect submissions that contain the same or similar errors. If such grouping is possible and can be automated, it could allow an instructor to generate feedback for a particular error once and have the system use that feedback every time a student submits a solution containing the same error. Thus, our second research question was, *"How many distinct logic errors exist in student submissions and how many submissions contain each one?"*

For the programming exercises that we examined, the number of distinct logic errors was found to be small enough to allow the instructor to easily deal with each error once. If the logic errors could be recognized automatically, the AA system could immediately send detailed feedback to any student submission, as long as the errors it contains have already been encountered. This would result in a substantial labor savings for instructors, as well as providing the benefit of offering rapid, detailed, well designed feedback to the students.

2. RESEARCH METHODOLOGY

The study was conducted in an undergraduate computer science course that introduces students to object-oriented programming in C++. The 61 registered students were assigned weekly sets of programming exercises. Each set contained, on average, 6 exercises. Weekly laboratory sessions were three hours long, with a teaching assistant available to help students debug their code. Upon completion of an exercise, students were to submit their solutions to our AA system. The system compiled and ran the student's code against a set of test cases, and provided binary ("Correct" / "Incorrect") feedback. There was no limit on the number of resubmissions allowed.

To answer our first research question we analyzed data collected by our system on each submission. For each programming exercise, we considered all students who made one or more incorrect submissions. We further separated this group of students into those who successfully arrived at a correct solution, those who gave up on the exercise, and those who resorted to plagiarism in order to obtain a correct solution. (A student solution was considered plagiarized only if it was *identical* to another student's solution.) In order to further assess the effectiveness of the provided binary feedback, we calculated the time each student took to correct an initially wrong solution. This is simply the time difference between the first and last submission, in cases where the last submission was correct. We made the assumption that if a student took a long time to arrive at a correct solution, then the student did not really benefit much from the feedback provided by the system.

In the second part of the study, we looked at the possibility of clustering incorrect submissions for a given exercise in such a way that instructor-generated feedback for one submission would be appropriate for all other submissions in the same cluster [3]. The intuition behind this approach arises from informal reports from educators indicating that many students tend to make the same mistakes, forcing the repeated delivery of the same feedback. Due to the large number of submissions we collected, we arbitrarily selected five exercises, and we manually examined each submission for each of these exercises, generating appropriate feedback for each submission. Submissions were then clustered based on the feedback that we had generated. In order for two submissions to be grouped into the same cluster, the same feedback had to be judged as equally appropriate for both. We then examined the number of clusters as well as the size of each cluster to determine whether or not it makes practical sense to cluster programs with similar errors together. We also examined the kinds of errors made, as well as their frequency of occurring.

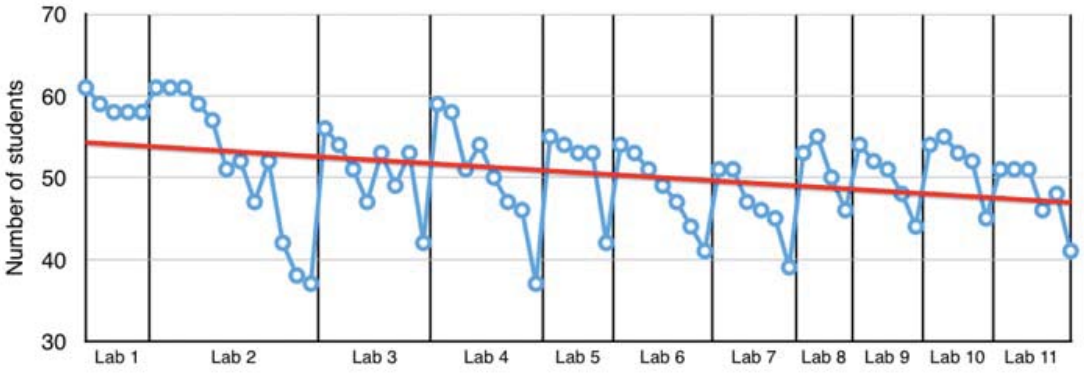

Figure 1. The number of students attempting each exercise

3. RESULTS

We first examined the number of students attempting each exercise, assigned in each lab session. Figure 1 shows these data in chronological order. The number of students attempting a given exercise declined steadily as the course progressed. This decline is even more pronounced across the exercises within each individual laboratory session. This suggests that the provided binary feedback was insufficient to reliably guide students to produce correct solutions in enough time to complete all assigned exercises.

Next, we computed average performance statistics on a per-exercise basis. As shown in Table 1, 50 of the 61 students in the class, on average, attempted each exercise. Of these 50 students, 24 submitted a correct solution on their first attempt. An average of 19 students failed each exercise, due to either not attempting it, giving up before producing a correct solution, or plagiarizing a correct solution. We have previously reported evidence that binary feedback contributes to these kinds of failures [4].

The other 18 students, on average, corrected their initially incorrect solutions. In order to further evaluate the utility of the provided binary feedback, we looked at the time it took each of these students to arrive at a correct solution, after an unsuccessful first attempt. Figure 2 shows four different brackets of time taken by students to fix initially incorrect solutions and the percentage of students that fell in each bracket. Of the students

providing an incorrect first submission, 43% corrected their program in 5 minutes or less. For these students, binary feedback appears to have been sufficient. Unfortunately, there were a large number of students who took much more time to arrive at a correct solution. Each laboratory session was 3 hours long, and there were 6 exercises, on average. This means that students who took more than 30 minutes per exercise would not have been able to complete all of the exercises during the session. This explains the sharp decline in exercise completion rates per laboratory session, observed in Figure 1.

Given these results, it is clear that our AA system was not very helpful for about half of the students. These students might have had fewer difficulties, however, if higher quality instant feedback on their programming exercises was delivered. To assess if such feedback might arise by classifying each incorrect solution into one of a small number of clusters, with each cluster manually provided with pedagogically appropriate feedback by a trained educator, we examined the distribution of errors that were made. We arbitrarily selected 5 exercises, and we clustered the incorrect submissions by grouping together similarly incorrect submissions, as previously described. Table 2 summarizes the results of this analysis. It is clear from these data that a large fraction of the errors made were shared by a large number of students.

Average number of students attempting an exercise	50
Average number of students not attempting an exercise	11
Average number of students with correct solution from first attempt	24
Average number of students who corrected an initially incorrect program	18
Average number of students who cheat to obtain correct solution	6
Average number of students giving up on an exercise	2

Table 1. Summary of average data on a per-exercise basis

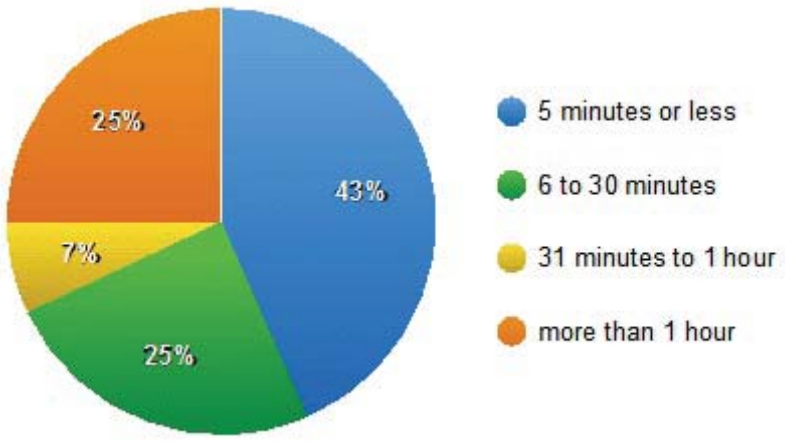

Figure 2. Distribution of time needed to correct an initially incorrect submission

Of the five exercises examined, the first was less challenging than the others. A total of 111 incorrect programs were submitted, but these contained only 4 distinct errors. The

other exercises exhibited 8-10 distinct errors, which was still substantially lower than the number of incorrect submissions. Each of these exercises had clusters of size 1, indicating that there were errors made by only one student. Despite this fact, the average cluster sizes are large enough to warrant further research into algorithms that can cluster similarly incorrect solutions to programming exercises.

Finally, we briefly discuss the kinds of errors found for these five exercises. We focus on logic errors, arising from submissions that compiled successfully. We found that our students were generally able to quickly correct syntax errors, given compiler output.

Exercise 1 involved reading a real number and writing out the area of a circle with that radius. The 4 observed errors were: hardcoded input (no reading), area variable declared as an integer, insufficient precision in the declared value of π, and superfluous output. The largest clusters were superfluous output (52) and defining π imprecisely (54), covering all but 5 incorrect submissions.

Exercise 2 involved counting the number of words in a text file specified as input. 12 of the submissions counted only the spaces, while 4 counted only the number of lines. 13 programs incremented word counters for a blank line, and 3 neglected to ever increment their counters. 6 submissions had an uninitialized word count variable, and 8 produced hardcoded output, providing the correct output for an example text file that was given. 18 programs hardcoded the file name instead of reading it, as required.

Exercise number	Incorrect submissions	Number of clusters	Largest cluster size	Smallest cluster size
1	111	4	54	2
2	82	10	18	1
3	73	11	19	1
4	28	8	15	1
5	26	8	13	1

Table 2. Summary of data from clustering analysis

Exercise 3 was a modification of Exercise 2, in which occurrences of a specific input word were to be counted in a file, rather than counting all words. 19 submissions produced superfluous output, and 8 hardcoded all or part of the input. 6 programs just output the correct answer for a provided example text file. Only 1 submission failed to deal with case sensitivity correctly. 4 programs did not consider punctuation symbols at the end of words. 12 submissions failed both with regard to case sensitivity and punctuation at the end of words.

Exercise 4 involved reading integers until a -1 is read. Programs were to output either EVEN or ODD for each integer except the -1. There were 15 submissions that produced output for the -1 terminator. 4 programs only processed the first number read.

Exercise 5 required the sorting of a list of integers, using any sorting algorithm. Many students tried to implement Bubble Sort, with the most common problem being an incorrect stopping condition for the inner loop of this algorithm.

The errors outlined above vary in how difficult they would be to automatically detect. In early exercises, the most common error is superfluous output, which is easy to detect. Hardcoding of input and output can also be detected fairly easily. More sophisticated exercises introduce difficulties in detecting specific errors, however. While a relatively small number of error clusters were found for the examined exercises, further research will be needed to automatically classify incorrect submissions.

4. CONCLUSION

We sought to assess the effectiveness of AA systems providing binary feedback on programming exercises. We found that about half of our students produced a correct solution for a given exercise, on average, either on their first attempt or shortly thereafter. Binary feedback AA systems seem to assist these students by simply alerting them to the presence of errors. Unfortunately, a large number of students could not leverage binary feedback to quickly correct their mistakes. Some of these students stopped trying. Others resorted to plagiarism. This group of students might benefit greatly from more detailed feedback, provided in a timely manner. There were not many students who fell in-between the two extremes of solving exercises quickly and struggling for a long time.

This finding encourages us to seek ways to provide rich and detailed instant feedback. This would be facilitated by automatically classifying submissions into error clusters. We found that a small number of such clusters typically arise. For the exercises examined, students made roughly 10 distinct errors. Providing rich feedback for only 10 error cases per exercise would result in substantial labor savings for instructors.

REFERENCES

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

[2] Beaubouef, T., Mason, J., Why the high attrition rate for computer science students: Some thoughts and observations. *SIGCSE Bull.*, 37(2):103-106, June 2005.

[3] Kyrilov, A., Noelle, D. C., Using Case-Based Reasoning to Improve the Quality of Feedback Provided by Automated Grading Systems. *E-Learning*, Lisbon, 2014.

[4] Kyrilov, A., Noelle, D. C., Binary Instant Feedback on Programming Exercises Can Reduce Student Engagement and Promote Cheating. *Koli Calling*, Finland, 2015.

[5] Woit, D., Mason, D., Effectiveness of online assessment. *SIGCSE Bull.*, 35(1):137-141, 2003.