# Using Case-Based Reasoning to Automatically Generate High-Quality Feedback for Programming Exercises

Angelo Kyrilov
University of California, Merced
5200 North Lake Road
Merced, CA 95343, USA
akyrilov@ucmerced.edu

## ABSTRACT

My research explores methods for automatic generation of high-quality feedback for computer programming exercises. This work is motivated by problems with current automated assessment systems, which usually provide binary ("Correct"/"Incorrect") feedback on programming exercises. Binary feedback is not conducive to student learning, and has also been linked to undesirable consequences, such as plagiarism and disengagement.

We propose a Case-Based Reasoning approach to utilize knowledge created by human instructors in order to automatically generate comparable responses for students that submit incorrect solutions to programming exercises. Such a system would offer significant labor savings for instructors, without sacrificing the quality of student learning.

Preliminary experiments have demonstrated the strength of our Case-Based Reasoning approach and its potential impact, especially in MOOCs. Further research is being conducted in order to refine the procedure and to evaluate it effect on student learning.

## 1. INTRODUCTION

Computer programming is becoming an essential skill in today's economic climate. This has led to significant enrollment increases for introductory Computer Science (CS) courses, as students from virtually all disciplines are required to learn programming. In order to cope with the increased workload, many CS educators rely on automated assessment systems for programming exercises.

Automated Assessment systems for computer programming exercises have been studied widely. [1] provides an overview of automated assessment approaches, and [7] studied the effectiveness of automated assessment on student learning. The authors found that systems which offer instant feedback and allow for multiple resubmissions are helping students to learn.

Some researchers are opposed to using such systems, mainly because of the poor quality of feedback they offer students. In many cases feedback is limited to a binary response ("Correct"/"Incorrect"). [2, 6] argue that in order for learning to take place, students who have generated incorrect solutions to a particular programming exercise, need to be given *guidance* by an expert programmer, and that simply pointing out the presence of an error is not enough.

We studied the effects of binary feedback on students and found that it increases their propensity to cheat on programming assignments and/or disengage from the course material [4]. A possible explanation for this is that since a binary response does not explain the reasons for failure, nor does it suggest a possible strategy to resolve the problem, students are often left with little choice but to cheat or given up on the exercise.

In [3], we proposed a Case-Based Reasoning approach to address the issues surrounding binary instant feedback. The idea is to use knowledge previously generated by human instructors in order to automatically build meaningful responses to incorrect programs submitted by students. In practice, such a system would have a signifiant impact in both traditional classroom environments as well as Massive Online Open Courses (MOOCs). We believe that automated feedback, comparable in quality to human-generated responses, will address motivation problems in MOOCs, which is expected to lead to increased completion rates. In regular university settings, the labor savings will allow instructors and teaching assistants to spend more time on activities beneficial to their students, rather than grading or debugging students' code.

The rest of the paper is organized as follows. Section 2.1 is an overview of our research, and a motivation for the chosen directions. Section 3 presents preliminary results, and highlights the potential contributions of this work. Section 4 outlines research questions that will be explored in future studies, and Section 5 contains concluding remarks.

## 2. RESEARCH TOPIC

### 2.1 Case-Based Reasoning

Case-Based Reasoning (CBR), first introduced by Schank [5], is a problem solving framework that uses past experiences to solve problems. Past experiences, referred to as a *cases*, are stored in a database, known as the *case base*. A single case consists of a problem description and a solution.

When a new problem, or a *query*, is encountered, the CBR system retrieves past cases whose problem descriptions are similar to the new problem, and uses the past solutions to generate instructions on how to solve the query. If executing the instructions does not lead to a solution of the problem, then the instructions are revised and evaluated again. Revisions may take place multiple times, until the solution generated by the system is accepted. At this point a new case, made up of the query and the accepted solution, is stored in the case base, making additional knowledge available for future queries. Due to its ability to create new knowledge in this way, CBR is considered a machine learning technique.

The CBR process can be summarized as the following four stages, illustrated graphically in figure 1:

1. **Retrieve:** Retrieve past cases that are similar to the query.
2. **Reuse:** The retrieved cases are used to generate a solution to the query.
3. **Revise:** The solution generated in the last step is evaluated and modified if necessary.
4. **Retain:** A new case, made up of the query and the solution are stored in the case base.

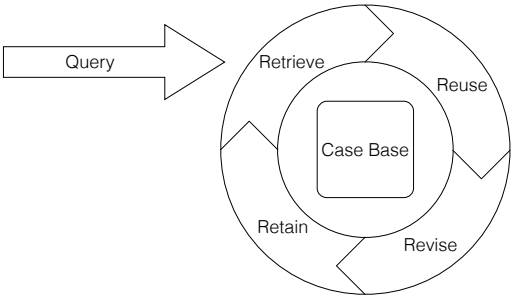


**Figure 1: The case-based reasoning methodology**

## 2.2 Proposed System

The automated assessment system we propose utilizes a Case-Based Reasoning approach to automatically assess computer programming exercises and provide feedback to students. We define a case to be a pair made of an incorrect computer program $P$, and instructor-generated feedback $F$. A computer program is deemed incorrect if it does not produce the expected outputs for a given programming exercise. Cases are therefore exercise-specific. Our case base is simply a collection of such cases.

For the retrieval stage, we need to define method of computing similarity between cases. Two cases $(P_1, F_1)$, and $(P_2, F_2)$ are said to be similar if $P_1$ is *similarly incorrect* to $P_2$. Two programs are similarly incorrect if they both contain the same bugs, therefore corrective feedback for one of the programs is equally appropriate for the other. In the reuse stage, we use the feedback retrieved at the previous step, without any modifications. This is possible due to the way we have defined the similarity metric for two cases.

The revise step, if necessary, will be performed by a human instructor. This is the way the system creates new knowledge. The revise procedure will be invoked if the student repeatedly submits an incorrect solution to the same exercise. This would suggest that the feedback offered by the system has not been helpful to the student. Once a correct solution has been submitted by the student, a new case is stored in the database. This case is made up of the original incorrect source code and the feedback that led to the submission of a correct solution. Figure 2 is a graphical representation of the proposed system.

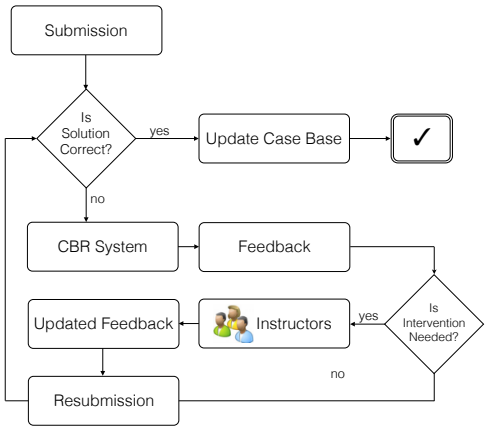


**Figure 2: A flowchart of proposed system**

## 2.3 Motivation

Previous research on Case-Based Reasoning has shown that the technique is most effective when similar problems are encountered often and when similar problems have similar solutions. Both of these conditions hold in the context of computer programming exercises. Indeed, CS educators often see the same mistake made by many students, and due to the asynchronous nature of laboratory sessions, the instructor is forced to give the same explanation to multiple students. The second condition, that similar problems have similar solutions holds true as well. If two or more students have all made the same mistake, they will all benefit from the same explanation. There could be multiple ways to explain the same mistake, and some students may find one explanation more beneficial than others. This is easily addressed by allowing the system to store multiple feedback comments per case, and present them sequentially upon unsuccessful attempts. The system can also keep track of the likelihood a particular feedback comment will lead to a successful resubmission and use this information to determine the order in which comments will be presented. Both conditions have been verified experimentally, with results presented in Section 3.

# 3. PRELIMINARY RESULTS

To test the soundness of our proposed system, we gathered student submissions from an undergraduate Computer Science course where students were required to complete pro-

gramming exercises on a weekly basis. Students uploaded their solutions to an automated assessment system that evaluated their correctness using unit testing.

The first research question we sought to answer was whether or not our proposed system was feasible. We randomly selected 5 exercises and extracted all the incorrect submissions for each one. We then manually clustered them according to their incorrectness. The results from this clustering procedure are presented in Table 1.

| Exercise Number | Incorrect Submissions | Cluster Count | Largest Cluster | Smallest Cluster |
|---|---|---|---|---|
| 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 1: Summary of clustering experiment**

It is clear from Table 1 that the same mistakes are made by many different students. This is indicated by the large values in the "Largest Cluster" column. In 4 of the 5 exercises we considered, there were mistakes committed by only one student, but small clusters are generally rare.

A more interesting and significant finding was that the number of clusters is relatively small compared to the total number of incorrect submissions. The average number of clusters is 8. This means that there are only 8 different mistakes that students are making, on average. This result is significant because an instructor with an empty case base will only need to grade 8 exercises by hand. The CBR system would be able to provide the appropriate feedback to every subsequent incorrect submission. The number of clusters is also not expected to grow with the number of students enrolled in the class. This is because the number of clusters is a function of the problem, not the number of students.

If the system scales well, it would enable MOOC instructors to provide corrective feedback to tens of thousands of students who have submitted incorrect solutions to programming exercises. This is likely to increase student engagement with the material and improve overall completion rates.

# 4. FUTURE WORK

In order to realize our system design, we need a reliable way to automatically detect similarity with respect to incorrectness between two programs. Our initial approach was to compute this similarity based on the unit tests. That is if two programs fail the exact same set of unit tests then they are deemed to be similarly incorrect. This is a reasonable first approach but it generates many false positives and false negatives. To ensure true scalability, the false matches need to be kept to a minimum. Methods involving static analysis of source code will likely need to be employed.

Further investigation of our scalability claims is also needed. More submission data would have to be analyzed and relationships between class size and number of clusters would need to be formally established.

Once the system has been completed, it should be deployed in a classroom and its effectiveness should be studied.

# 5. CONCLUSION

My research is focused on improving the quality of instant feedback generated by automated assessment systems for programming exercises. Many instructors are using automatic grading systems that are limited to providing binary feedback, which has been shown to hinder student learning and lead to plagiarism and disengagement.

We propose a Case-Based Reasoning approach to designing an automated assessment system for programming exercises capable of instantly delivering high-quality feedback, comparable to guidance a human instructor might provide to a struggling student. The system uses feedback previously generated by human instructors and delivers it to students who make similar mistakes to ones seen before.

This is an effective technique since the same mistakes are made by many different students and there are relatively few distinct mistakes. This translates into significant labor savings for instructors and teaching assistants. With our system in place, an instructor will only have to address a specific problem once. All subsequent occurrences will be handled automatically by the system.

Further research is currently being conducted on finding a reliable metric for similarity with respect to incorrectness of computer programs. Several static analysis techniques are being explored. Attempts are also being made to formalize relationships between the class size and the number of unique errors that can be made on an exercise. We postulate that for reasonably sized programming exercises, the number of unique errors will stay low even in MOOC environments where class sizes can be in the hundreds of thousands.

# 6. REFERENCES


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

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

[3] A. Kyrilov and D. C. Noelle. Using case-based reasoning to improve the quality of feedback provided by automated grading systems. In *Proceedings of the International Conference on E-Learning*, pages 384–388, 2014.

[4] A. Kyrilov and D. C. Noelle. Binary instant feedback on programming exercises can reduce student engagement and promote cheating. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, Koli Calling '15, pages 122–126, New York, NY, USA, 2015. ACM.

[5] R. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, New York, NY, USA, 1982.

[6] G. N. Walker. Experimentation in the computer programming lab. *Inroads*, 36(4):69–72, 2004.

[7] D. Woit and D. Mason. Effectiveness of online assessment. *SIGCSE Bull.*, 35(1):137–141, Jan. 2003.