Enhancing Automated Feedback in On-Going Assignments

Huanyi Chen
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
huanyi.chen@uwaterloo.ca

Paul A.S. Ward
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
pasward@uwaterloo.ca

ABSTRACT

In programming courses, test-based automated feedback systems often face a limitation: instructors cannot effectively enhance feedback during ongoing assignments. While a passing test case may adequately indicate that certain rubric criteria have been met by a student's program, failure can arise from many reasons. When a test case fails due to reasons not previously coded for, it can create confusion. This tends to divert a student's focus from genuine learning towards guessing the test. A more effective approach would be for the system to notify instructors when a test fails, rather than automatically returning a "test case failed" message. Instructors could then diagnose the cause to provide an initial "human-in-the-loop" feedback. Subsequently, this feedback can be coded into the test suite. This method enables the improvement of feedback during an on-going assignment. In this poster, we propose an approach to achieve this objective and introduce a supporting tool.

ACM Reference Format:
Huanyi Chen and Paul A.S. Ward. 2024. Enhancing Automated Feedback in On-Going Assignments. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3626253.3635571

1 PROBLEM STATEMENTS

The automated feedback system is a vital component in programming courses. Instructors rely on it to deliver timely feedback to students on a large scale. While a passed test case often reasonably indicates that a student's program satisfies certain required rubric items, the converse is not typically true. When a test case fails, the cause can be unforeseen. For example, if a test writer assumes the execution result of the student's program is always numeric, a line to cast the result into integers might be used. However, if a student forgets this rule, the test case will fail due to invalid casting. Given that this is unexpected, it is unlikely that this test can provide correct feedback. This can be particularly challenging for students who are already unsure about their solutions.

Adding to the complexity, instructors usually find it difficult to modify or correct the assessment code during an ongoing assignment. This is because changes to the assessment code could lead to students receiving inconsistent feedback for the same solution submitted at different times. Consequently, instructors often postpone addressing these issues until the assignment concludes, implementing any improvements in subsequent offerings of the same assignment [1].

Just as assessments often require modifications, assignments too frequently need revisions, especially newly designed ones. As a result, new assessment code is introduced, and the aforementioned issues re-emerge.

The problems can be roughly summarized as:

(1) **Unclear failure cause**: The assessment code offers limited insight into the actual reason behind a failed test case.
(2) **Inconsistencies**: Modifying the assessment code can lead to inconsistent feedback messages.

In this poster, we propose an approach that allows instructors to intervene when unexpected situations arise during assessment. This approach also empowers instructors to enhance the assessment code without waiting for assignments to conclude. We open-source a supporting tool to fulfill the idea. We hope for it to find broader adoption.

2 PROPOSED SOLUTIONS

Figure 1 shows the overview of our solutions.

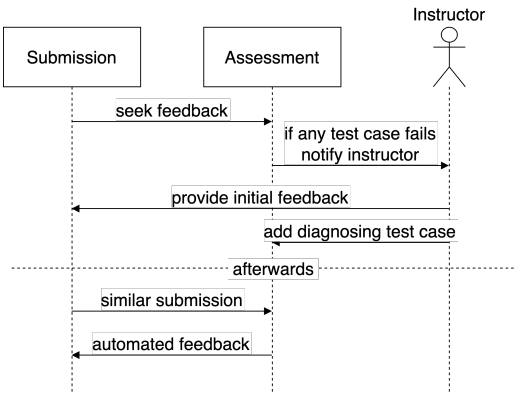

Figure 1: Enhancing automated feedback procedure

2.1 Unclear failure cause

The rationale behind our solution for this problem is that, a passed test case indicates all code of the test case has executed as expected, while a failed test case can fail due to unforeseen causes, which may or may not be the location anticipated by the test case writer. Thus, the specific reason for such failures might not have been pre-coded.

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA.

ACM ISBN 979-8-4007-0424-6/24/03.
https://doi.org/10.1145/3626253.3635571

To address this, we consider the assessment code should only report if a program is correct, *i.e.*, all test cases passed, but it should not report if a program is incorrect, *i.e.*, some test cases failed. For the latter, it notifies instructors and tell students to wait for human-in-the-loop feedback.

Students anticipate feedback to correct misconceptions. Therefore, the assessment code should help diagnosing the root cause of test case failures. Instructors, upon inspecting a failed submission, can introduce a "diagnosing test case" to pinpoint the failure cause when it passes.

This iterative process continues until all root causes can be identified by diagnosing test cases.

2.2 Inconsistency due to assessment code changes

We propose that instructors should always translate test case results into actionable feedback to reduce the cognitive burden on students when interpreting test case outcomes. If a primary test case fails, instructors should reference the corresponding passed diagnosing test case to provide feedback.

Although instructors will not automatically be notified of positive test outcomes, they can detect false positive results through manual inspection. Since we do not disclose the outcomes of individual test cases to students, we can offer feedback such as, "*Your solution seems sound! However, we encourage you to think about potential edge cases that our current evaluation might miss.*" This approach fosters critical thinking among students, and helps students to separate feedback and grades. Receiving perfect feedback may not ensure a solution is perfect, however, it may indicate that the solution is on the right track and likely to gain a reasonable grade.

Nonetheless, we believe it is preferable not to introduce code changes that invalidate previously correct solutions in primary test cases, as this can cause insecurity among students, leading to a surge in resubmissions.

Modifications for above cases should be implemented after the assignment is concluded. In making these modifications, the approach used in Haldeman et al. [1] could be one of the options.

For false positives within diagnosing test cases, instead of immediately modifying the test case, we might consider adjusting the feedback to include, "*...or another possible issue could be ...*".

As for addressing false negatives, since we prevent students from seeing any feedback when any test case fails, therefore, we can make corrections as soon as we identify an issue.

3 THE SUPPORTING TOOL

We have developed a tool[1] designed to help instructors incorporate our ideas into their courses. The current version requires instructors to create test cases using the popular "pytest" testing framework. Although primarily built for Python, it can be configured to assess assignments of other programming languages as well. For instance, we are currently using it for database assignments.

The tool needs instructors to provide feedback messages with diagnosing test cases. If an automated feedback is available, the feedback is used; if not, an email is sent to the instructor. What are included in the email are configurable. It also allows the instructor to generate a draft AI feedback and include it in the email.

Currently, we consider a feedback message might need to be associated with one or more diagnosing test cases. Therefore, our tool supports matching a set of diagnosing test case outcomes to a feedback message. However, we do not have a strong rationale behind the assumption.

The tool is a command-line application that can be distributed, installed, and executed in various environments. Although the tool does not offer much assistance in writing assessment code, it provides a common structure for individuals to develop additional features for their assessment code and potentially share them, beyond what we have implemented, such as sending emails and generating AI feedback.

4 DISCUSSION

The quality of assessments should be the most essential component in automated feedback systems; however, this has been rarely discussed. Our approach outlines a procedure to improve the quality of assessments deterministically.

We are currently using the tool in a database course. In particular, we will primarily use it to assess students' Structured Query Language (SQL) queries. Beyond this, we have also developed encoding techniques to encode Relational Algebra (RA) and Entity Relational (ER) modelling solutions. We will explore how the tool helps for those questions as well.

We integrate our tool into an automated system where students submit their work. We found the tool is very helpful. For example, many students encountered a similar runtime error. We quickly added a diagnosing test case to detect it and provide an interpretation.

We had to spend time on monitoring the emails when it was close to the assignment deadlines. However, we believe instructors would inevitably have to invest the time if the automated feedback were not helpful anyways (which was also our experience).

Although we have found the AI can generate fairly accurate feedback on some types of mistakes, whether we can directly send back the AI feedback is still unclear (even though we have done so in cases such as syntax errors).

Given the AI generated feedback, it might seem to shift the focus of instructors from adding new tests to reviewing the AI feedback before sending it to students. However, our experiences show that, it is unclear what prompts should be used to generate high-quality AI feedback. We observed that sometimes the AI feedback leaks the canonical solution. However, we are unsure how to disallow it even though we have tried adjusting the prompts.

We will evaluate the efficacy of the tool by comparing students' performance with data from previous terms, wherein the tool was not deployed. We anticipate observing a marked improvement in students' proficiency in the relevant learning objectives.

REFERENCES

[1] Georgiana Haldeman, Monica Babeş-Vroman, Andrew Tjang, and Thu D. Nguyen. 2021. CSF: Formative Feedback in Autograding. *ACM Transactions on Computing Education* 21, 3, Article 21 (May 2021). https://doi.org/10.1145/3445983

[1] https://github.com/h365chen/socassess