Nudging Student Learning Strategies Using Formative Feedback in Automatically Graded Assessments

Lucas Zamprogno
lucasaz@cs.ubc.ca
Department of Computer Science
University of British Columbia
Vancouver, Canada

Reid Holmes
rtholmes@cs.ubc.ca
Department of Computer Science
University of British Columbia
Vancouver, Canada

Elisa Baniassad
ebani@cs.ubc.ca
Department of Computer Science
University of British Columbia
Vancouver, Canada

Abstract

Automated assessment tools are widely used as a means for providing formative feedback to undergraduate students in computer science courses while helping those courses simultaneously scale to meet student demand. While formative feedback is a laudable goal, we have observed many students trying to debug their solutions into existence using only the feedback given, while losing context of the learning goals intended by the course staff. In this paper, we detail two case studies from second and third-year undergraduate software engineering courses indicating that using *only* nudges about where students should focus their efforts can improve how they act on generated feedback. By carefully reasoning about errors uncovered by our automated assessment approaches, we have been able to create feedback for students that helps them to revisit the learning outcomes for the assignment or course. This approach has been applied to both multiple-choice quizzes and automated programming assessment. We have found that student performance has not suffered and that students reflect positively about how they investigate automated assessment failures.

CCS Concepts: **• Social and professional topics → Computing education**; **Computer science education**.

Keywords: assessment, software engineering, autograder

ACM Reference Format:
Lucas Zamprogno, Reid Holmes, and Elisa Baniassad. 2020. Nudging Student Learning Strategies Using Formative Feedback in Automatically Graded Assessments. In *Proceedings of the 2020 ACM SIGPLAN SPLASH-E Symposium (SPLASH-E '20), November 20, 2020, Virtual, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3426431.3428654

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SPLASH-E '20, November 20, 2020, Virtual, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8180-2/20/11…$15.00
https://doi.org/10.1145/3426431.3428654

1 Introduction

Automated assessment techniques have been widely used in computing education [1, 9, 15]. These assessments are often used to help enable classes scale to larger sizes [2, 14, 24]. One downside of automated assessment strategies is that they must be carefully designed to avoid students trying to randomly permute their answers to satisfy the automated system in a way that would not be possible with human assessment.

From the student's perspective, one benefit of automated assessments are that they shift grading from a means for assigning a grade to a mechanism for providing useful feedback so that they can assess their learning and determine how to improve their understanding [13]. Providing meaningful formative feedback in an automated setting requires careful balancing between giving complete low-level feedback for incorrect answers (which may encourage random permutation) with intentionally vague feedback (which may not provide enough direction to encourage learning).

This paper investigates two specific research questions:

RQ1 Can providing students with higher-level formative feedback nudge students towards revisiting learning outcomes?

RQ2 Can higher-level formative feedback enable better reuse of instructional materials?

To investigate these questions, we describe two case studies where we have applied our nudge-based feedback approach for automated assessment systems. The goal of these approaches is to guide student learning strategies for internalizing automated feedback towards course learning objectives. We use the term *nudge* because the feedback our graders produce does not tell the student specifically what they did wrong in the assessment. Instead, it provides hints about areas upon which to focus further study.

The first case is a second-year required software construction course (SC) that is taken each year by over 900 students. This course uses mastery learning based micro-assessments [25] to help students evaluate their learning. The assessments are provided through a series of multiple choice 5-minute quizzes that can be retaken until successful. Instead of returning fully-marked quizzes, we returned *only* hints to give the student a starting point for preparing for their re-take assessment.

The second case is a third-year required software engineering course (SE) that is taken each year by over 600 students. This course uses a large-scale development project that is assessed using an automatic marking environment (called AutoTest) that the students can invoke during development to gauge the correctness of their solution. Instead of returning all test failures, we returned high-level feature-based feedback to nudge students towards examining the project specification and their own test suites, rather than trying to debug their solution into existence by making many rapid changes in hopes of causing a test to pass.

While these case studies were from different assessment domains, applying these approaches forces course staff to think concretely about what is being taught to the student so feedback can emphasize what knowledge or content they should revisit. Staff must also develop a mapping from failures to learning objectives the students should be nudged towards.

This paper contributes a discussion of mechanisms for nudging student response to feedback delivered by automatic grading systems, an approach and evaluation of outcome-oriented feedback for multiple choice software engineering exams, and an approach and evaluation of outcome-oriented feedback for programming assignments.

2 Case Study 1: Hint-Based Quiz Feedback

We applied our approach to provide formative, hint-style feedback from multiple choice micro-assessments to a second year required software construction (SC) course. Our initial goal in applying the approach was to nudge students into revisiting the course learning objectives and materials when they got questions wrong, rather than quickly guessing alternative answers to superficially pass the quiz.

2.1 Quiz Feedback Background

Multiple Choice Quizzes lend themselves to automatic grading. Typically frameworks that deploy multiple choice quizzes online, or automatically, return the correct and incorrect annotations for each question, along with explanations for why each question was right or wrong. EdX, for instance, has a facility to provide individual feedback messages for a correct or incorrect answer, but always within the context of annotating the original quiz. [1]

Other mechanisms for personalised feedback are also being explored. OnTask, for instance, allows individualised messaging to students [19]. MessageStudentsWho, a Canvas based tool, also allows individualised messaging [12]. These facilities are all at quiz-level granularity, meaning that the messages can comment on whole assessments, but will not, without the context of the original quiz, return individually corrected answers to the students. Messages will look more like "*You obtained 9/10 on your last quiz. You have another quiz next week! Remember to do practice modules A, B, and D, which you have not yet completed!.*"

[1] https://edx.org

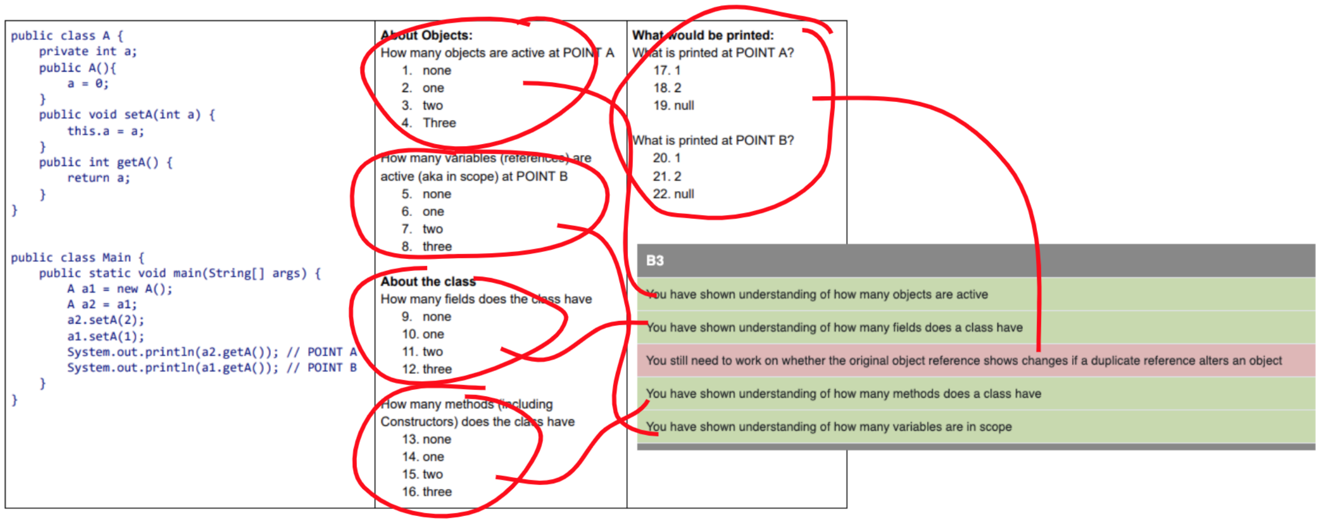
Clariana, Ross, and Morrison investigated the effectiveness of different feedback strategies for multiple-choice questions [7]. Students were given a multiple choice test followed by a feedback condition, then later retested on the same material. There were five feedback conditions. Two involved no feedback, one being not getting the question set at all prior to the retest. The three feedback conditions were knowledge of correct response (KCR) in which you are told the correct answer, delayed KCR, and answer until correct (AUC) where students can repeat the question until they get the right answer. All forms of feedback showed benefits over no feedback, with AUC showing more benefit as the retest questions became more different from the initial test.

Later work by Clariana and Koul produced a meta-analysis of multiple-try feedback (MTF), where students are able to make multiple attempts to answer incorrect questions, compared to other forms of feedback [6]. They found that MTF was less effective when test questions were verbatim and pulled directly from the instructional material. However MTF was more effective when the test questions required a higher-order level of reasoning about the material. They suggest that this means that MTF functions differently than other feedback, and promotes increased semantic learning at the cost of directly recalling content.

Butler and Roediger studied the ability of feedback to increase the positive learning effects of testing [5]. Participants took a multiple choice exam, followed by either no feedback, feedback immediately after each question, or feedback at the end of the test. Participants were then retested after a one week delay. They found that both forms of feedback reduced the likelihood that previously presented options were erroneously remembered as the correct response. Both forms of feedback also increased the proportion of correct responses on the retest, with the delayed feedback condition having a stronger effect in this area.

In the realm of hint generation, Price et. al. [20] looked at hint use and abuse, and found that the quality of generated hints led to more use of hints, but if the hint quality was very high, an overreliance on hints was identified. In this case a high-quality hint might be one that is overly specific, leading to students overfitting to learning the test instead of the concepts. Marwan et. al. [18] found that hints improved performance in highly related tasks, and that forcing students to explain the hint in their own words was more facilitative of learning than hints alone.

For all styles of feedback studied above, there is evidence that they enhance student learning, but all of these modes of feedback are linked specifically to a problem set rather than dislocated from it. This means that in each case, students might be tempted to focus on the specifics of what they did wrong (or right), without necessarily re-contextualising their learning more broadly.



```
public class A {
    private int a;
    public A(){
        a = 0;
    }
    public void setA(int a) {
        this.a = a;
    }
    public int getA() {
        return a;
    }
}

public class Main {
    public static void main(String[] args) {
        A a1 = new A();
        A a2 = a1;
        a2.setA(2);
        a1.setA(1);
        System.out.println(a2.getA()); // POINT A
        System.out.println(a1.getA()); // POINT B
    }
}
```

About Objects:
How many objects are active at POINT A
1. none
2. one
3. two
4. Three

How many variables (references) are active (aka in scope) at POINT B
5. none
6. one
7. two
8. three

About the class
How many fields does the class have
9. none
10. one
11. two
12. three

How many methods (including Constructors) does the class have
13. none
14. one
15. two
16. three

What would be printed:
What is printed at POINT A?
17. 1
18. 2
19. null

What is printed at POINT B?
20. 1
21. 2
22. null

B3

You have shown understanding of how many objects are active

You have shown understanding of how many fields does a class have

You still need to work on whether the original object reference shows changes if a duplicate reference alters an object

You have shown understanding of how many methods does a class have

You have shown understanding of how many variables are in scope

Figure 1. Mapping from Quiz Questions to Feedback Messages shown to a student. Students never see their quiz again, unless they book an appointment with an instructor to go over it. They are not allowed to take photos or notes of their original quiz, and must leave it behind at the end of the meeting.

2.2 Relating Failures to Learning Objectives

We introduced a hints-only approach for providing feedback to students on their mastery quizzes in a second year software construction course. Our approach was to provide feedback *messages* to students, rather than returning back to them the quiz they took. If they wanted to visit the quiz again, they could come to instructor office hours and go over it, but they were not allowed to take photos of it or write any notes about the specific questions. This was done to promote quiz reuse, as they are time consuming to create, and we were hoping to reuse them across semesters.

The technical approach for accomplishing the quiz feedback was described in a prior paper [3]. Each selection in the students' multiple choice quizzes was associated with a course learning outcome or skill using a mapping file.

2.3 Representing Formative Feedback

Students would receive their quiz grades back on a dashboard that would show either a red "*you need more work on...<the skill>*" message when the student had answered that question any of the related questions incorrectly, or a green "*you have shown mastery of ...<the skill>*" message if the student had answered all of the questions related to that skill correctly. A depiction of this is shown in Figure 1.

Because we were deploying these quizzes in a mastery learning model, the students were able to retake quizzes until they passed. Some quizzes were required quizzes, forcing students to retake them to pass the course.

2.4 Study Design

After one full semester of deploying this technique (in which we gave roughly 10K quizzes), we noticed anecdotally that students were asking far more general questions on the online class forum[2] after failing quizzes. This suggested that students were unable to focus solely on the specific details of what they did wrong, and instead were being forced to revisit the higher level concepts (at the skill level) to study for their retake quiz. Also, even though they had the option available, only a handful (around 5) came to visit their quizzes in the cohort of over 600.

We decided to perform a survey asking students about their study strategies for the retake, specifically about what preparation steps they took. We wanted to know whether the feedback messages helped guide their study, and/or to what extent they revisited the concepts of the topic. We also wanted to know whether students felt hopeless because the quizzes were not returned.

We asked students:

> "If you have to retake a quiz, what is your process for preparing for the retake? What do you check/read/try/ask/etc?"

2.5 Analysis and Results

258 students who had previously had to retake quizzes responded to our question. We read all the responses, identifying key terms or phrases. We then coded students' responses by automatically looking for keywords in the text of their responses. We then manually removed false positives where students were not using the terms to convey a process. For instance, students typically used the word "understand" in the context of a statement such as "*I would then try to understand the concepts.*" If a student said "*I just don't understand why we can't see our original quiz*" then this would not be considered a true match for the term "understand". We also

[2] https://piazza.com

looked for false negatives. For instance a comment such as "*check the red highlighted sentences*" would not have been automatically coded as being associated with the feedback dashboard, but there is no other interpretation for that comment, so it would be counted as an instance of a student explicitly identifying the feedback messages as part of their process. In cases where the associations were ambiguous, we did not add them. We refer to the resulting categories from this analysis as "strategy elements".

Watch Videos (150/258 students). A large number of students mentioned re-watching the videos associated with each topic. Because our course is underpinned by one or more videos per topic, students are able to select videos or parts of videos to watch, based on the concepts they want to revisit. To identify if students chose that strategy, we looked for students who mentioned the terms "watch" or "video" in their response.

Students said things such as:

— "I rewatch the relevant videos, and try to review what concepts I struggled with the first time."
— "Go through the videos and take some notes to understand the concepts. Try extra practice problems."

Do Practice (121/258 students). A large number of students explicitly mentioned words related to re-doing practice assignments. Practice assignments in the course consisted of the pre-class work, the practice questions that accompanied the videos, and the in-class activities.

Study More (96/258 students). A moderate to large number of students mentioned words related to revisiting the topic for the sake of conceptual understanding. We looked through the statements for the words "concept" and "understand", and looked for active studying words, such as "review" and "read".

Check Feedback Messages (53/258 students). A moderate number of students explicitly mentioning checking for the topics they got wrong in the quiz dashboard as part of their process. To assess this we initially looked for the terms "check", "handback" (which is the name of the underlying system we use), and "dashboard" (the name of the tool). Verbatim examples stated by students included:

— "seeing what I did incorrectly in handback, and reviewing the material"
— "check the grades handback for feedback"
— "check the concepts that are marked red in the score dashboard"
— "i would check what i messed up, watch the videos on the concept"

Get Help (25/258 students). A moderate number of students mentioned getting help in response to failing a quiz. The kinds of help mentioned were instructor office hours, and three students mentioned asking a friend for help.

No Plan (10/258 students). A small number of students had no clear plan for studying for their retake quiz.

They supplied feedback such as:

— "I failed one quiz due to being pressured by the time. I just told myself to calm for the retake and I got 100"
— "Sadly I don't have the time usually to prepare for a retake, I have a very busy workload so I do not prioritize it"

Negative Comments (4/258 students). Several students did report having study strategies, but also indicated they were not happy about having to have them. They made comments such as:

— "TAs don't help with anything"
— "Well we can't really check much. The system of telling us that we need to retake it without showing us our errors is rather ridiculous and flawed. I've retaken 1 quiz maybe 4 times because I don't know what I'm doing wrong despite reading the 'hints' on the grade page"

Even though these comments are infrequent, it does give us, as educators, pause to revisit the quiz hints to ensure they are giving students enough information to succeed. Typically in cases where students have failed a quiz multiple times, we reach out to the student to go over the material with them, and get them further support.

Visiting Their Quiz (0/258 students). It was interesting to us that no students mentioned asking to see their quiz again, though students were informed that this was an option. Only a few students in the class asked to see their quiz. It is possible none of them were in the response group, or they had not considered it a strategy.

Overlapping Strategies (162/258). The results tell us that almost all students had at least one strategy, though many included statements that were aligned with multiple strategies. Most students had one or two strategies with some having three strategies and a few with four. A small number had no strategies. 90 students indicated one strategy, 101 indicated two strategies, 49 students indicated three strategies, and 12 students indicated four strategies.

Strategy Combinations. We also looked at what the combinations of strategies were. There were 26 strategy combinations. We gave each strategy a keyword, and tagged each student with the strategies they mentioned. We then counted the frequency of the combinations.

The two most popular strategies by far were practicing using the course resources combined with watching the videos, and just watching videos.

Anecdotal Experience in Office Hours. We have noticed that questions posed by students about their failed quizzes are different in nature than when they are returned their original quizzes. Students will ask questions like "what's the difference between a checked and unchecked exception?" as opposed to saying "why did I get this question wrong?".

Interestingly, we still give back midterm exams using the traditional approach, and the same students who had broad and contextualised strategies for their quizzes still arrive in our offices with detailed questions about their individual midterm questions, and individual grading elements for specific questions.

2.6 Implications of the Findings

Only 65 of the 258 respondents said explicitly that they would check their feedback hints as a starting point for revision. This may be because the feedback appears at the bottom of the results page, and they have to scroll to find it. The thing they initially see is just a red or green circle indicating success or failure for their quiz. It's possible that the students who did not explicitly mention the hint messages were only looking at whether they'd failed the quiz, and then reviewing the entire topic from there. At the very least, the feedback messages were not dominant in the minds of most of the students.

The dramatic difference in the tonal presentation of students asking for help with passing their next quiz, versus students fighting for individual marks on their midterm (even though the material will certainly be revisited on the final, in a different form), is highly encouraging, and suggests that providing feedback messages has had a profound impact. It bears noting that students did not carry over their conceptual revision approach to their midterm reflection. It would have been quite impressive if students had changed their way of reflecting on assessments *generally* when looking at how they did on an exam, but in this case we only see the study strategy effect when the more tempting target of focus, the individual question grade, is not available.

Ultimately, it is interesting and encouraging that almost all students did have a coherent study strategy for revision, with many just being happy to re-watch all the videos and re-formulate conceptual understanding.

3 Case Study 2: Hint-Based Programming Feedback

The second case study took place in a required third year software engineering course. It includes a large TypeScript-based project split into multiple checkpoints that each take three weeks to complete. As the student teams work, they can submit their in-progress solutions to AutoTest which runs a test suite against their code and returns formative feedback. In this section we describe how we applied our approach to try and nudge students to a more self-reflective approach to developing their solution.

3.1 Programming Feedback Background

Programming tasks are well suited to formative feedback, as software is usually developed iteratively. Students will devise an approach to the problem at hand and then attempt an initial implementation of their idea, which will frequently be flawed in some way. Perhaps there was a side case that was not considered, but their idea is fundamentally correct, or maybe some mechanism was misunderstood and the approach as a whole is flawed. In both cases, the solution needs to be refined. Receiving quick, formative feedback helps tighten this development loop and encourage students to make ongoing improvements.

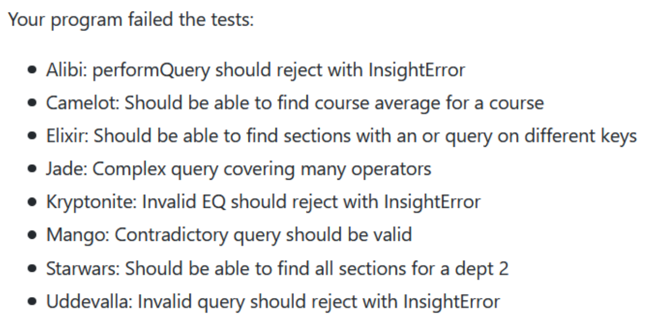
Automatic Grading. Automatic graders have long been used for programming assignments [14]. Several factors account for automatic grading being chosen over traditional manual-assessment approaches. Automatic assessment reduces time burdens for course staff and decreases assessment latency for students. This quick turnaround opens up a system for students to iterate with multiple submissions. If grading is handled by a computer, then there is no human time cost to allowing students to receive repeated feedback over the course of an assignment [11]. This also frees instructors and teaching assistants to spend their time providing more hands-on and personal assistance for their students and decreases some barriers associated with large-course management. Automatic assessment is one key approach being used to handle the increased demand for computer science courses [2].

Ihantola et. al. performed a literature review of recent automatic assessment systems [15]. Their review focused on the different features frequently found across assessment tools including target language, testing framework, re-submission policy, facilitating manual assessment, and specialisation.

Mumuki is an online open-source coding tool supporting 17 languages [4]. This tool supports two assessment categories: functional correctness via unit tests, and expectations which are constraints on implementation details. Introducing Mumuki into CS1 and CS2 courses yielded a statistically significant reduction in dropout rates.

Web-CAT is an automated grading tool that is designed to be a flexible and language agnostic system [10]. It can mark submissions on test validity, completeness (coverage), and code correctness, where each of these steps is customizable by course staff. Feedback is given to students in the form of color coded bars for test pass rate and code coverage, with detailed numerical information presented below.

Tailored Automatic Grader Feedback. Some prior work has examined tailoring automatic grader feedback to more effectively facilitate course learning goals.



Your program failed the tests:

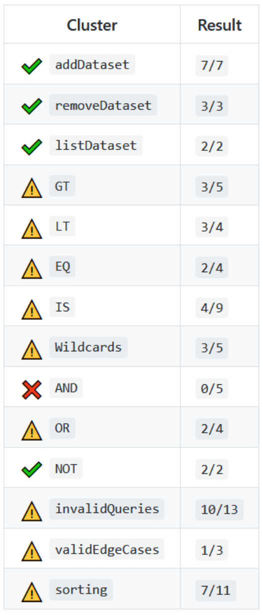
- Alibi: performQuery should reject with InsightError
- Camelot: Should be able to find course average for a course
- Elixir: Should be able to find sections with an or query on different keys
- Jade: Complex query covering many operators
- Kryptonite: Invalid EQ should reject with InsightError
- Mango: Contradictory query should be valid
- Starwars: Should be able to find all sections for a dept 2
- Uddevalla: Invalid query should reject with InsightError

Figure 2. Original test-based automated feedback. In this mode, students get feedback on all of their failing tests.

Kandru created a plug-in for Web-CAT to provide more intelligent and goal oriented feedback for students [17]. The system categorizes errors as coding errors, behaviour problems, testing issues, and style problems. Each category has its own subcategories containing specific feedback. This high-level approach provides clear areas of focus for improvement and is similar to our approach. However we elect to not return details of the underlying failures with the intention of guiding students to reflect on the weak areas of their submission and avoid them trying to spot-fix their problems without having to step back and reflect on the issue.

Haldeman et. al. use a system where assignments, tests, and feedback are designed around a course knowledge map [13]. Once an assignment has collected a number of student submissions, submissions are batched by what tests were failed into 'buckets'. These buckets are then refined and subsequently matched with the concept from the knowledge map, where many buckets may correspond to the same concept. From this point on each concept is assigned a hint which can be returned to students who have failing tests. Our approach differs in two main ways: First, our clusters (analogous to buckets) can be created before deploying to students by identifying functional components of the project. Second, we return the status for all clusters on every test run, but do not provide direct hints, requiring students to re-evaluate the specification or write more tests.

Singh, Gulwani, and Solar-Lezama present a tool for providing low-level program corrections for MIT's Introduction to Programming course [21]. This system can identify errors in student submissions and automatically provide a set of steps the student should follow to fix their program. It does so using a reference implementation and searching a space of millions of corrections to find the fix with a minimal amount of changes. Similarly, TraceDiff uses program synthesis to try to debug student submissions and provide personalized feedback [22]. These approaches differ from ours in that they focus on code fix suggestions so students can repair their code, where our approach focuses on higher-level problems to encourage students to step back and re-evaluate the problem specification and develop their solution.



Cluster	Result
✓ addDataset	7/7
✓ removeDataset	3/3
✓ listDataset	2/2
⚠ GT	3/5
⚠ LT	3/4
⚠ EQ	2/4
⚠ IS	4/9
⚠ Wildcards	3/5
✗ AND	0/5
⚠ OR	2/4
✓ NOT	2/2
⚠ invalidQueries	10/13
⚠ validEdgeCases	1/3
⚠ sorting	7/11

Figure 3. Formative feedback given to students for their submission. Four clusters have all associated tests passing, one has all associated tests failing, and the other clusters are partially complete.

3.2 Encouraging Introspection With Failures

We have applied our nudge-based approach to a third-year required undergraduate class in which students complete a moderate-sized programming project (approximately 2,500 lines of code) over the course of the term. The project is split into three checkpoints and is completed in teams of two.

Each checkpoint is graded using two sets of automated tests: the first is a public test suite from which students can receive limited formative feedback while the checkpoint is in progress; the second is a private test suite the results of which are withheld until after the checkpoint is due. Students commit code at will and are able to invoke AutoTest once every 12 hours. In this way our test suites act as if the students are following test-driven development (TDD) [23] as the tests exist prior to their implementation.

Prior to this work, AutoTest feedback flowed naturally from how unit testing works in practice: a list of all test case failures was given to the students. This output is shown in Figure 2. The test names had descriptions ranging from fairly descriptive (e.g., "Should be able to find sections with an or query on different keys") to non-descriptive (e.g., "Invalid query should reject with InsightError"). Descriptive feedback sometimes led to adverse reactions or interpretations that would inhibit students from making progress. Specifically, when presented with a list of failures, students had trouble determining what features were common between the failures, or prioritized low-importance edge cases over more important parts of the checkpoint. A secondary downside was that students would often debate the meaning of the feedback on the class forum[3], instead of inspecting the checkpoint specification.

As a part of this case study, we modified AutoTest feedback to cluster any test case failures with the functional requirements of the checkpoint. In this way, the approach simplifies the output by only telling the students how they are doing on each of the major features of the checkpoint as shown in Figure 3. This format is more outcome-oriented by mirroring components of the specification that students must implement. Rather than positioning the automated test suite as TDD, we instead strongly encourage the students to think of the clusters as user acceptance tests where a customer is validating the high-level features of the system. We expected that providing students with this clustered feedback would encourage them to think about their solution in reference to the assignment, instead of getting hung up on specific details of the underlying assessment.

3.3 Relating Failures to Learning Objectives

In this section we discuss our approach for clustering feedback based on unit test failures. One important aspect of this approach is that unit test failures can be associated with multiple learning outcomes. If a test requires multiple features implemented to pass, it will appear in each of those features' clusters. Two steps are required to implement this: the learning outcome clusters must be identified, and a failure in a student solution must be mapped into these clusters.

Our approach for reducing feedback is to group failures into learning outcome clusters. This first requires that we examine the checkpoint specification and determine what the clusters should be. For instance, some parts of the specification, like individual query operators, are typically developed discretely from others, whereas other project functionality, like query validation, are more cross-cutting and are shared between several features. Another consideration is how many clusters to show the students: while the goal was to reduce the feedback to students to nudge them to look at the specification, we had to balance that with not giving specific enough feedback to guide them. Ultimately we identified 14 clusters our project, depicted in the left column of Figure 3.

Next, we had to determine which tests in our test suite corresponded to each functional cluster. Rather than manually assigning tests to clusters (which may be straightforward with smaller simpler tests but is harder for larger more complex tests), we chose an an approach that was inspired by mutation testing, which has shown that small changes (mutations) to a program can simulate real faults [16]. To do this, we manually introduced mutants into our reference solution for parts of the code we knew implemented each learning outcome cluster and observed which tests failed for each learning outcome mutation.

[3]https://piazza.com

Choosing mutations can take some thought, for instance to generate a cluster for sorting results we had to change our sort implementation to sort randomly, because simply disabling sorting missed some tests due to our default result order matching the intended sorting rule in some cases. While it is tempting to use traditional mutation testing to break the software first and then identify the functionality of the clusters that appear as a result, this will generate a large number of highly specific clusters for any project of reasonable size. For our project, running an automatic mutation test pass on a single file led to 77 distinct clusters, which did not meaningfully map to the specification features.

After running our test suite for each chosen cluster, we manually verified that the tests assigned to each cluster seemed reasonable, for instance by ensuring that edge case tests were classified correctly. Ensuring each test is present in a cluster is crucial because otherwise clusters will not provide a complete picture of a student project's completeness, and resulting grades. The main risk for tests being left uncategorized comes with edge cases that do not fit cleanly into the normal functionality or common error scenarios for the program. When deciding on what broader clusters of functionality are displayed to the students, small but important details may not fit naturally in any category and mistakenly be left out.

While our edge case tests ended up in at least one other cluster, we still created an additional cluster for "valid edge cases"[4]. This required tests to be selected for the cluster manually, as there is no singular code location to handle edge cases, nor are they strictly related in functionality or behaviour. One purpose of this cluster is to provide a nudge for students to expand their thinking about other features if they are left with a low number of test failures with no singular cause. We also wanted to highlight that edge cases represent a small portion of the overall assessment, and encourage students to focus their efforts on core functionality first.

3.4 Representing Formative Feedback

There were two platforms we had to support for displaying cluster results. The student view is provided via GitHub commit comments, and only displays the clustered form of the feedback. The interface presented to students was iteratively designed through informal consultation with senior TAs who had recently taken the course. The clusters were represented with a simple Markdown table layout showing a `pass/total` value for each cluster. We also included an emoji for each cluster representing that all, some, or no tests passed, so students could understand their results at a glance. We ordered the clusters manually by the order we thought

[4]In the context of our project, these could be cases such as a queries that are well formed according to our grammar, but may be unexpected or unintuitive.

best for addressing them, although this was not explicitly stated to the students.

Figure 3 shows an example of the clustered test feedback as seen by students. This is a partially complete solution where a student's solution passes all tests involving data sets, as well as the `NOT` query operator. No tests which involve `AND` pass, and all other clusters have at least some failing tests. In this case the students should realise that their `AND` implementation is broken or missing, and that completing core functionality of that feature is the next important step. This case also shows the challenge of providing feedback for tests that cross-cut many features from the specification. Noting that the students are at 3/4 for their `LT` cluster, a reasonable assumption is that the remaining failure could be due to an `LT` filter being inside an `AND` which is failing, and not the `LT` implementation itself. With the prior approach of a simple list this problem is even less transparent.

There is also an administrative view for course staff provided by Classy, our course management system[5] which is integrated with AutoTest. In this system we allowed TAs to toggle between the new clustered view and a traditional view in which individual tests are visible. The clustered view was useful for gaining a quick overview of a team's solution, although the TAs often used the list-of-test failures when they needed more insight.

3.5 Analysis and Results

We simultaneously applied this approach to two sections of our course with 384 students in total. All students were enrolled in the experimental condition which clustered the results to avoid giving some students unfair advantages over others; students only had access to clustered results and this condition was used for the duration of the project. When compared to the prior term (380 students) students overall performed similarly. The median score of the students who had the clustered results increased by one percent with the average dropping by one percent. There was also a small dip in perfect scores.

There was a notable difference on the class forum in both post quantity and quality. Overall there was a drop from 3.0 posts per student to 2.2 posts per student with the introduction of the clusters. The largest qualitative change, likely related to the drop in total post count, was the absence of student posts requesting direct hints for specific tests. Previously, one common post archetype was the form "*I am failing Test X, any hints?*" Once the individual tests were hidden behind clusters, this type of post no longer works. A few posts of the form "*I am failing one test in cluster X*" appeared, but they were much less frequent compared to the prior posts.

At the end of each checkpoint, a survey was sent out to all students including a question regarding their experience when receiving feedback from the automatic marking environment. The main question with regard to the AutoTest feedback, and the only one we analyzed, asked:

> "If you got a negative result back from AutoTest, what did you do to resolve the issue? For instance – what do you read, try, code, run, ask, etc. and roughly in what order?"

Of the 331 responses, 237 contained meaningful content.[6] To analyze these, one author performed an open card sort [8] to identify common recurring themes in the feedback from these 237 students. Five high-level categories emerged from this analysis:

Examining Personal Tests (146/237 students). Part of the project is encouraging students to understand the benefits of TDD and create their own comprehensive local test suite. Historically, students would often rely on the automated feedback as their primary method of gauging the correctness of their solution. When the clustered feedback was given instead, students noted that their own tests gained importance and referred to them more: "*AutoTest was rarely the thing giving meaningful feedback (our tests were a lot more descriptive and easier to get feedback on).*"

Creating New Personal Tests (87/237 students). Even with a prior assignment to develop tests for our specification, often students only spot new edge cases that could arise after trying to make an implementation and seeing it break. For example, students would augment their test suite after learning their tests, which they previously thought were complete, needed additional work: "*We'd ask ourselves what the thing failing is supposed to do, and what we expect it to do in every situation we could imagine. Then we'd make tests based off of those expectations and confirmed if it was doing what we thought it should.*"

Reviewing Own Code (61/237 students). A common response for students when given feedback that something they wrote does not function correctly is to read over the code responsible, visually tracing program features to try and spot bugs. This can sometimes be a backup technique when no local tests fail, which means running their code will not highlight the issue. One student stated that they primarily rely on their own tests, however: "*When we had a problem with autobot, we reread the code, had the other partner look at the code, and then asked for help from a TA.*" This indicates a good understanding of what code elements are likely responsible for the error, but without a hypothesis for what the underlying assessment might be that is not covered by their local tests.

Examining the Specification (58/237 students). In the past, students would often focus their efforts on making the

[5]https://github.com/ubccpsc/classy

[6]Many comments were either too general to understand their context or were too specific to project-specific implementation challenges.

individual tests mentioned in feedback pass, losing sight of the original specification which should guide the construction of their system. With the nudge-based feedback this shifted; for example, one student noted that they would, "*Read the specification again to see if I missed anything. If I find something suspicious I'll add a test to see if it passes.*" This was re-enforced by another student who stated that they would "*ask ourselves what the thing failing is supposed to do, and what we expect it to do in every situation we could imagine.*"

Focusing on the Most Failing Cluster (12/237 students). While this did not emerge as one of the common categories, we expected focusing on notable weak points before working on mostly-working features to be one of the most effective ways to make progress using the feedback. This strategy was most succinctly captured by a student who reflected that "*I looked at the status of the test clusters, and focused on the 'low-level' clusters (e.g. GT, LT, IS) which were not all passing, because those could have cascading effects on other tests clusters.*"

Reflecting on these responses, the prevalence of students who addressed feedback returned by the testing framework by examining their own unit test suites and strengthening them was gratifying to see. Simultaneously, we were disappointed with how few students mentioned specifically looking at their implementation associated with the failure cluster with the highest proportion of failures (which would have been our expected optimal strategy for making the quickest progress). Ultimately, we saw many students engaging (broadly) in fault localization through their references looking for "*that specific area [associated with the feature]*" while examining their own implementation in response to our failure feedback.

4 Discussion

In this section we examine how the feedback returned to students could nudge them to reflect on their submissions thoughtfully. We also examine how thinking about failure mapping helped us to improve the tests we were using to evaluate the student submissions. We also discuss threats to validity. Finally, we end the paper with a discussion of the impact automatic assessment feedback can have on the strategies students use for addressing failures.

4.1 Facilitating Assessment Reflection (RQ1)

RQ1: Can higher-level feedback nudge students towards learning outcomes? For both case studies we found evidence of students more commonly revisiting both the course learning outcomes and project specifications. Notably, decreased feedback did not notably decrease student understanding or performance.

In addition to our positive findings in Section 2.5 and Section 3.5, connecting learning objectives to assessed elements also improved the formation of the overall assessments.

In the SE programming project course, we discovered an imbalance between our use of different operators in our queries when we mapped the learning objectives to test failures. For instance, initially our test suite `AND` operator cluster had many more tests than our `OR` operator cluster. These should be similarly challenging to implement and there is no reason one should be valued more than the other, so we altered the tests to compensate. We also noticed an imbalance in tests between the public and private test suites, with our `NOT` cluster containing no tests in the private suite. Again we shuffled tests between suites to remedy the imbalance. The value of this balancing comes from wanting to making sure our assessment was aligned with our learning objectives, and that students receive appropriately granular feedback across objectives.

Our anecdotal observations of the changes in online class forum use in SE tells us that students strategies did change after the introduction of the nudge-style feedback. The way students respond to their quiz results is starkly different from the way they respond to their midterm exam result (by arguing over the grading scheme for individual questions), suggesting a fundamental change in strategy.

We saw this same pedagogical benefit in the SC course quiz feedback approach. By assigning feedback hints to specific quiz questions, we were able to see the conceptual coverage of each quiz. For instance in the "B3 quiz" shown in Figure 1, we can see that the hints are mostly balanced, but the multiple-reference hint is associated with more questions than each of the other feedback messages. We were also able to better identify what important concepts quizzes were not testing. In the B3 quiz, for instance, we can tell without reading the quiz that there were no lists involved in the assessment, since none of the hints relate to lists storing objects. These conceptual annotations helped give a level of oversight of the pedagogical assessment strategy that we had not predicted would be present when we initially adopted the nudge-based approach.

4.2 Assessment Artifact Reusability (RQ2)

RQ2: Can higher-level feedback increase material reuse? Using higher-level formative feedback enabled better between-term reuse in the SC course as quiz-based materials were not clearly available to the students; similarly, in the SE course the automated test suites could more easily be reused as they were also harder to directly decipher from the limited feedback.

Reusability is an important aspect of automatic assessments due to the overhead required in creating them. In the the SC course, we have over 20 quizzes with 4 isomorphic variants for each. For the SE course, we have over 300 unit tests that have been carefully crafted over a multi-year period. In both cases, being able to reuse these artefacts enables staff to invest a level of time and commitment that would otherwise be untenable were they only available to be used once.

Providing more abstract feedback to students increased the reusability of the assessment artefacts themselves. In the SC course, not pointing out specific incorrect questions meant their copied 'value' to the students was decreased, and cheating became much more difficult. Additionally, the abstract feedback itself could be reused across multiple questions for which the original question is isomorphic with no decrease in assessment value.

Not providing specific test feedback in the SE course enabled us to reuse the test suites across multiple course offerings as the students do not actually know what the individual tests were in each cluster.

4.3 Threats to Validity

Both of the case studies described in this paper were performed at the same R1 research university, potentially limiting generalisability. While the course staff for each case study were independent, they did coordinate on assessment measure and pedagogical goals. All of the students in the studies were Computer Science Majors and both courses were required for the program. While they used different languages (Java for software construction and TypeScript for software engineering), we do not believe language selection has an impact on the feedback design for these courses. As such, we see these results likely being generalisable at least within SC/SE, and potentially more broadly.

While we have a series of metrics that enable us to compare both student cohorts (before and after our feedback intervention), in terms of internal validity we did not ask students for their problem solving strategies for the original feedback design; having this information would certainly improve our ability to determine whether the feedback alone (which was the only real change between the course instances) was the means for changing their learning strategy. Another threat to internal validity is the impact of the *quality* of feedback messages on their utility to students. We did not examine feedback messages in isolation, controlling for degree of specificity. As such, our findings may be impacted by whether students could understand the messages themselves, but because we did not control for this, we are unable to assess the extent of this effect. It is reasonable to assume that our messages were adequate, that is they were neither optimal, nor completely flawed, and so it follows that we are assessing a somewhat average case of feedback message creation, and association with assessment elements. Finally, while grades are not scaled in either course, and we expect student ability to be relatively constant across these large courses, we have not controlled for the overall student abilities between offerings of the courses. While the majority of students provided written feedback for both studies, there may have been a potential self-selection bias given feedback was not required.

Both feedback deployments were applied in situations where students could have multiple tries for success. It is not clear whether high-level nudges would be appropriate in situations where no retries are available, and we would feel hesitant about employing nudges in such a situation.

5 Influence on Student Strategy

In both courses, we saw that a majority of students formed concrete study strategies to prepare for their subsequent attempts at success. In the SE course, the prevalence of starting from the failing tests was strong – students predominantly integrated checking failure messages from the automated assessment framework as the starting point of their revision. In the SC course, a smaller percentage of students explicitly stated that they started by looking at the messages, but no students stated that they included booking a meeting for the sake of visiting their quizzes in their strategy. There were only four negative comments about the nudges, suggesting that, on balance, students believe not getting back their original quiz was acceptable.

Anecdotally, we are able to see a difference between the strategies formed when students are provided hints only, and when they are provided their original tests back with feedback embedded into them. In the SE course we administer multiple choice midterms, and return students their original tests, with explanations for their individual selection's correctness annotating their test. When provided their original test, students litigate the correctness of individual questions, instead of revisiting the material as a whole. Students have similar investment in re-learning the material, because our final exams are cumulative, and include the concepts that were present on the midterm. Still, students focus on why they lost points for specific questions (which certainly will not appear in the same form on the final exam), rather than stepping back and trying to identify gaps in their general understanding.

Conversely, in the SC course, we use the same automatic grading tool as the SE course, but with specific test failure messages, as opposed to nudge-based messages. Students in office hours for the SC course discuss individual tests, and how to pass them, and, in our recollection, seldom bring up the specification other than to complain that it is not telling them how to pass specific tests.

Ultimately, these two case studies seem to indicate that it is possible for hint-only feedback to not hurt observed or perceived student performance, and instead has the ability to nudge students towards positive, more holistic, and contextual learning strategies.

References

[1] Kirsti M Ala-Mutka. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education (CSE)* 15, 2 (Jun 2005), 83–102. https://doi.org/10.1080/08993400500150747

[2] Sylvia Alexander, Una O'Reilly, Pat Sweeney, and Gerry McAllister. 2002. Utilizing automated assessment for large student cohorts. *Engineering Education and Research–2001: A Chronicle of Worldwide Innovations,* (2002).

[3] Elisa Baniassad, Alice Campbell, Tiara Allidina, and Asrai Ord. 2019. Teaching Software Construction at Scale with Mastery Learning: A Case Study. In *Proceedings of the International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET).* 182–191. https://doi.org/10.1109/ICSE-SEET.2019.00027

[4] Luciana Benotti, Federico Aloi, Franco Bulgarelli, and Marcos J. Gomez. 2018. The Effect of a Web-based Coding Tool with Automatic Feedback on Students' Performance and Perceptions. In *Proceedings of the Technical Symposium on Computer Science Education (SIGCSE).* 2–7. https://doi.org/10.1145/3159450.3159579

[5] Andrew C. Butler and Henry L. Roediger. 2008. Feedback enhances the positive effects and reduces the negative effects of multiple-choice testing. *Memory & Cognition* 36, 3 (Apr 2008), 604–616. https://doi.org/10.3758/MC.36.3.604

[6] Roy B. Clariana and Ravinder Koul. 2005. Multiple-Try Feedback and Higher-Order Learning Outcomes. *International Journal of Instructional Media* 32, 3 (2005), 239 – 245.

[7] Roy B. Clariana, Steven M. Ross, and Gary R. Morrison. 1991. The Effects of Different Feedback Strategies Using Computer-Administered Multiple-Choice Questions as Instruction. *Educational Technology Research and Development* 39, 2 (1991), 5–17.

[8] Juliet M. Corbin and Anselm Strauss. 1990. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology* 13, 1 (March 1990), 3–21. https://doi.org/10.1007/BF00988593

[9] Christopher Douce, David Livingstone, and James Orwell. 2005. Automatic Test-Based Assessment of Programming: A Review. *Journal on Educational Resources in Computing (JERIC)* 5, 3 (Sep 2005). https://doi.org/10.1145/1163405.1163409

[10] Stephen H. Edwards. 2003. Improving Student Performance by Evaluating How Well Students Test Their Own Programs. *Journal of Educational Resources in Computing (JERIC)* 3, 3 (Sep 2003). https://doi.org/10.1145/1029994.1029995

[11] Xiang Fu, Boris Peltsverger, Kai Qian, Lixin Tao, and Jigang Liu. 2008. APOGEE: Automated project grading and instant feedback system for web based computing. In *Proceedings of the Technical Symposium on Computer Science Education (SIGCSE).* 77–81. https://doi.org/10.1145/1352322.1352163

[12] Julie Gregg, Melissa Diers, and Analisa McMillan. 2018. Hidden Treasures: Lesser Known Secrets of Canvas. (2018).

[13] Georgiana Haldeman, Andrew Tjang, Monica Babeş-Vroman, Stephen Bartos, Jay Shah, Danielle Yucht, and Thu D. Nguyen. 2018. Providing Meaningful Feedback for Autograding of Programming Assignments. In *Proceedings of the Technical Symposium on Computer Science Education (SIGCSE).* 278–283. https://doi.org/10.1145/3159450.3159502

[14] Jack Hollingsworth. 1960. Automatic Graders for Programming Classes. *Communications of the ACM (CACM)* 3, 10 (Oct 1960), 528–529. https://doi.org/10.1145/367415.367422

[15] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the Koli Calling International Conference on Computing Education Research.* 86–93. https://doi.org/10.1145/1930464.1930480

[16] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In *Proceedings of the Symposium on the Foundations of Software Engineering (FSE).* 654–665.

[17] Nischel Kandru. 2018. Intelligent Goal-Oriented Feedback for Java Programming Assignments. (Jul 2018). http://hdl.handle.net/10919/83947

[18] Samiha Marwan, Joseph Jay Williams, and Thomas Price. 2019. An Evaluation of the Impact of Automated Programming Hints on Performance and Learning. In *Proceedings of the Conference on International Computing Education Research (ICER).* 61–70. https://doi.org/10.1145/3291279.3339420

[19] Abelardo Pardo, Kathryn Bartimote-Aufflick, Simon Buckingham Shum, Shane Dawson, Jing Gao, Dragan Gaševic, Steve Leichtweis, Danny Liu, Roberto Martínez-Maldonado, Negin Mirriahi, et al. 2018. OnTask: Delivering Data-Informed, Personalized Learning Support Actions. *Journal of Learning Analytics* 5, 3 (2018), 235–249. https://doi.org/10.18608/jla.2018.53.15

[20] Thomas W. Price, Rui Zhi, and Tiffany Barnes. 2017. Hint Generation Under Uncertainty: The Effect of Hint Quality on Help-Seeking Behavior. In *Artificial Intelligence in Education*, Elisabeth André, Ryan Baker, Xiangen Hu, Ma. Mercedes T. Rodrigo, and Benedict du Boulay (Eds.). Springer, 311–322. https://doi.org/10.1007/978-3-319-61425-0_26

[21] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. 2013. Automated Feedback Generation for Introductory Programming Assignments. In *Proceedings of the Conference on Programming Language Design and Implementation (PLDI).* 15–26. https://doi.org/10.1145/2491956.2462195

[22] Ryo Suzuki, Gustavo Soares, Andrew Head, Elena Glassman, Ruan Reis, Melina Mongiovi, Loris D'Antoni, and Björn Hartmann. 2017. TraceDiff: Debugging unexpected code behavior using trace divergences. In *Symposium on Visual Languages and Human-Centric Computing (VL/HCC).* 107–115. https://doi.org/10.1109/VLHCC.2017.8103457

[23] Ayse Tosun, Oscar Dieste, Davide Fucci, Sira Vegas, Burak Turhan, Hakan Erdogmus, Adrian Santos, Markku Oivo, Kimmo Toro, Janne Jarvinen, and Natalia Juristo. 2017. An Industry Experiment on the Effects of Test-Driven Development on External Quality and Productivity. *Empirical Software Engineering (ESE)* 22, 6 (Dec. 2017), 2763—-2805. https://doi.org/10.1007/s10664-016-9490-0

[24] Chris Wilcox. 2015. The Role of Automation in Undergraduate Computer Science Education. In *Proceedings of the Technical Symposium on Computer Science Education (SIGCSE).* 90–95. https://doi.org/10.1145/2676723.2677226

[25] Tobias Wrigstad and Elias Castegren. 2017. Mastery Learning-Like Teaching with Achievements. *ArXiv* abs/1906.03510 (2017).