Your Test Cases Are Lying to You: Validating Student Software Tests Against Reference Implementations to Reinforce Specification Alignment

Abstract

In intermediate programming courses, students often work on large, multi-week programming projects with complex, multi-page specifications. Complex specifications provide opportunities for students to misunderstand requirements, leading to programs that pass student tests but fail reference tests, even if student tests have high coverage metrics or other markers of potential success. In such a situation, it is not possible for a student to find a "bug" in their program since it is internally consistent with their understanding of the requirements. As a consequence, students often attend office hours or use other help-seeking course resources to understand why their code fails hidden reference tests. This can become a major burden on teaching staff, and can lead students to ask for details about the reference tests that is not appropriate for staff to provide.

To help students better check their understanding of project specifications, we augmented our automated feedback system. The new feedback notifies students when their test cases fail against the reference implementation. This is important because it indicates that student tests assert a claim that does not align with the requirements as implied by the reference implementation.

We analyzed how students' use of this system impacted their performance on project grades. The correctness of the students' final submissions was measured on three projects across four semesters: two pre-intervention (Spring 2024, Fall 2024) and two post-intervention (Spring 2025, Fall 2025). Pairwise Mann-Whitney U tests indicated that submissions on projects from semesters using the test validation system generally achieved higher reference test pass rates than those from semesters without the system. Several comparisons showed statistically significant differences ($U \approx 49{,}600$-$104{,}400$, $p < 0.001$, $N > 350$). Effect sizes ranged from 0.16 to 0.40, indicating a small to moderate positive effect between the use of the validation system and improved correctness outcomes. Initial usage indicated that adoption was low as an optional mechanism, but subsequent projects required using the tool to ensure more consistent engagement. This paper presents the challenges and successes of deploying this test validation plugin for large programming projects and offers recommendations for instructors seeking to scaffold student testing practices to provide more robust test-adequacy criteria.

Introduction

Large, multi-week programming assignments are a staple of intermediate computer science courses. Specifications for this type of assignment can span multiple pages and include all of the relevant information to complete the project. While these detailed specifications can contribute to creating a real-world expectation for projects, they also create challenges for students [1]. These challenges are often related to the student's understanding of a project specification, or more specifically, their misunderstanding, and can be indicative of issues surrounding their comprehension or decomposition skills relevant to understanding a project task [2]. Addressing these conceptual or structural misunderstandings is integral to a student's ability to construct a correctly working program.

Many academic programming contexts ask students to write their own test cases to check their programming assignments. Software testing has long been used to encourage students to reflect on their code [3]. However, current strategies to influence students to create rigorous test suites often result in tests that embody the same misunderstandings as their code [4]. In addition, students only test to the level that they are required, focusing on test-adequacy criteria instead of adequate testing for the problem itself [5]. In particular, a common pattern students follow is creating a test case, then running that test case on their code to get an output. They will then use that output as the comparison for the test, creating a test case that achieves high coverage, but has no relationship to program correctness. Students then generate false confidence that their program is behaving as intended, while hidden instructor reference tests continue to fail. These gaps highlight the need for better validation of student test cases for large programming projects.

The students in this intermediate programming course predominantly consist of juniors who have taken multiple programming courses before this, yet many still struggle to write comprehensive tests without assistance. Students often encounter a situation where their tests are internally consistent with their understanding of the project specification, but out of sync with the actual specification. The only way out of this closed loop is to get outside information. But program autograders are not typically a good tool for detecting these misunderstandings since they will not share their reference tests.

We describe an intervention that explores this idea that students are not creating suitable tests without some extrinsic motivation or validation [5]. We provide an additional feedback section in the automated feedback system that shows students their "Test Validation Results", meaning the results of running a subset of their tests (those that meet an interface provided to them) against an instructor-provided reference implementation. By providing this feedback, we hope to reduce the dependence on TAs as oracles, allowing students to answer their own "what should the program do in this case?" questions through the automated system. To evaluate the intervention of providing this new feedback, we combine a quasi-experimental comparison of student submission quality (as defined by passing hidden instructor reference tests) between offerings of the course with and without the test validation feedback and a within-semester analysis of how students respond to the new feedback mechanism. We also conducted informal, informational interviews with students that indicated they found test validation helpful, but only after significant education on what the service actually provided. Throughout the lifespan of the projects, students indicated confusion as to why they were failing test validation despite their tests passing locally. Through

additional discussion of testing and an explanation of the system, students began to realize that test validation could help them answer questions about ambiguities in the project specification, thereby leading to faster feedback while also reducing load on TAs.

We begin by discussing relevant, related work in computer science education. We then describe the details of the intervention and the evaluation of the system. In the final sections, we assess threats to validity and summarize our conclusions.

Related Work

Students' understanding of specifications and their ability to design effective test cases are critical components of programming success, yet they are often overlooked or underdeveloped in traditional computer science curricula [6, 7]. Previous work has shown that, despite many courses requiring students to implement their own unit tests, students frequently experience misunderstandings and embody them in their test cases [8, 5]. Researchers have contributed several approaches to addressing this issue including better unit testing support, misconception-targeted interventions, and improvements to automated feedback systems. This section will summarize prior work in these three areas as they all relate to the proposed intervention.

Unit Testing: Students are often introduced to unit testing, a fundamental skill that computer scientists use to create better programs through validating their ideas about how their program should behave. Shin and Kazerouni analyze students' testing process and discover that students are more motivated by program coverage over problem coverage and that our current model of feedback on student testing practices may not accurately encourage students to test their programs well [5]. As a result, the authors call for a "requirements-first" approach to testing rather than the current "implementation-first" mindset [5]. Bai, Smith, and Stolee find that not only do students not recognize what makes a unit test good, they also often test only happy paths, situations where the programs behave in a common or expected manner [7]. They also found that students do not realize they may have difficulty understanding problem specifications and often write tests that do not match the problem specification. Other research seeks to broadly understand the behavior of developers as they write tests for code and can inform the behavior of students as well [9, 10]. In courses where students are required to write unit tests, Mansur et al. found that migrating from code coverage to mutation testing helped students write better tests and simplify their programs [11].

Misconceptions in Student Tests: Wrenn and Krishnamurthi contributed the idea that students can get frustrated when they do not realize they are actually solving the wrong problem. They assert that testing alone is not enough, that student written test cases often can embody the same misconceptions as their solution, as they were both most likely developed with the same misconception [4]. Further work by Prasad et al. created a system that mimics mutation testing (introducing faults into programs and validating that tests catch these faults) by requiring students to create examples of the problem that are verified against various implementations provided by the instructor where these implementations are generated from an analysis of misconceptions [12]. This work was then expanded again by Prasad et al. showing that incorrect submissions that students provide can be analyzed and clustered to provide useful misconceptions

to test future students with [13].

Automated Feedback Systems: Buffardi and Edwards note that writing tests can be a difficult task and students may default to using an automated grading system as an oracle and neglect their own tests [14]. They also note that students seem to neglect writing their own tests and only seek to perform the least amount of work to satisfy the automated grading system. This can lead to test cases that only validate the behavior of the student's program instead of test cases that validate the expected behavior of the assignment.

Spertus and Jimenez have also contributed an automated feedback tool that supports "cross-testing" [15], where student tests are evaluated on their ability to correctly catch faults seeded in hidden reference implementations while correctly passing valid implementations. These two works approach the idea that student tests must be validated to gauge their understanding of the specification, because just having students write tests is not enough.

Research Questions

We sought to address the following research questions.

RQ1: To what extent do students adopt the test validation system? For the Spring 2025 semester, test validation was turned on for all students, but in order to gain useful feedback from the system, students had to write assertions in the designated test file. We investigate how frequently students used the system throughout the semester.

RQ2: Does test validation use result in increased project correctness? We measure correctness as defined by instructor reference tests and evaluate if students using the system are submitting more correct solutions.

Methods

Participants

This work investigates project submission data from students enrolled in a post-CS2 Data Structures and Algorithms course (that we term CS3). The course was given at Virginia Tech, a large public R1 university in the United States. Students worked on four projects per semester in Java where they implemented one or more intermediate-to-advanced data structures (hash tables, binary trees, external sorts, or spatial data structures like quad trees). Students were given approximately three to four weeks to work on each project. Specification documents described the project requirements, an overview of the project, requirements, reference material, and recommendations for design considerations. These specification documents are typically three to four pages containing the relevant material along with boilerplate about programming standards and rules about standard library use. Students were given minimal starter code, often decreasing in size from Project 1 to Project 4 within any given semester. In Fall 2025, this starter code began including an interface for project methods (including insert, delete, dump, etc.) but students were otherwise free to design their own solutions. These methods were for a "database" container class and did not restrict students' implementations of the underlying data structures, though it is important to note that they were graded on the quality of their design choices. In addition to the

document, students were given significant guidance for what constituted a good design for the project.

Our data include submissions from four sections each in four semesters of this CS3 course. The intervention was first deployed for Project 4 in Fall 2024. Data from the eight sections in Spring 2024 (Projects 1, 2, 4) and Fall 2024 (Projects 1, 2) were used as a control group, and data from the eight sections in Spring 2025 and Fall 2025 were used as a treatment group with the test validation plugin, as well as Project 4 from Fall 2024. Note that Project 3 from all four semesters focuses on external sorting where students can always self-validate whether their solutions are sorted, and reference tests instead focus on performance, as correctness is implicitly required. As these projects do not use the validation plugin, we exclude them from our study.

For the student usage analysis indicated by **RQ1**, we analyzed the usage patterns of 330 students who submitted solutions to all three projects using test validation in the Spring 2025 semester. This usage data allowed us to examine patterns of adoption over time. For the project correctness analysis indicated by **RQ2**, we analyzed 5,349 final project submissions across all four semesters. This population included students who submitted at least once to any one of the 12 assignments in the study context. This submission data allowed us to investigate project correctness before and after the intervention.

Intervention

To help students validate their own understanding of the project specification, we augmented our automated feedback system, Web-CAT [3], with a plugin for "Reference Test Validation." This new plugin runs certain student unit tests against an instructor-provided reference implementation and provides feedback on the correctness of these tests from the perspective of the reference implementation (which hopefully mirrors the project specification). As these projects are intended to be open-ended from a design standpoint, with a significant portion of points dedicated to students' design choices, some compromises with how the system could be implemented were made.

As the system has evolved, the selected tests that the plugin runs have changed. For the Spring 2025 semester, the plugin ran tests using an input/output style that validated the program through print representations of data structures and data. This version of the plugin required the tests to be placed in a specific file, "ProblemSpecTest.java", that was then run against the reference implementation. This allows students to write solutions that differ from the reference solution that may have differently named methods, or even different classes.

During the Fall 2025 semester, in an attempt to improve our reference testing, we changed the programming projects to require that students write to an interface. Reference tests are therefore written to call methods from the interface. Figure 1 depicts a screenshot of the feedback that students get from the validation system, notifying them that some of their tests failed along with the error messages. Requiring students to implement the interface ensured their submission and the reference implementation's methods matched names, parameters, and return values. This allowed us to expand the plugin to validate any test that matched this interface in a specified testing file, "ExampleProjectNameTest.java", where students had been trained to write tests in previous programming courses.

If a student has a thorough grasp of the project specification and spends time developing good tests, the tests should all pass when run against the reference implementation. If a student has some misunderstanding about what format the output should take, this will appear as a test case failure against the reference implementation using test case validation (if they wrote a test case for that situation). Students commonly take a short cut and simply copy their own output into test cases. Their misconceptions are now built into their test cases as well as their code. While the reference test cases failing should indicate to students that something is wrong, they often do not understand how their code can have an issue if their tests pass. Test case validation will illuminate that there is an issue with their own test cases, since their faulty test cases will fail on the reference implementation. This has the additional benefit of providing students with a clear place to start debugging since they now have a test case that explicitly triggers a bug.

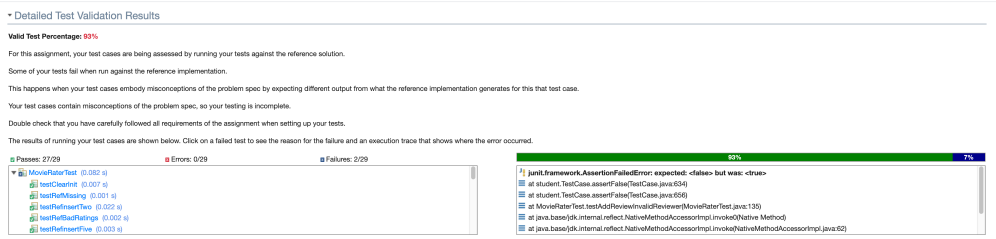

Figure 1: Validation results for a submission where the student had a misconception related to accepting or rejecting an input based on the requirements of the specification. This test failure lets the student know that even though their test passes on their code, the reference implementation expects different behavior for that case.

Data Analysis Procedure

Across the four semesters studied, there were eight unique projects, with four of the projects having been repeated (with or without slight modifications) between the control and treatment groups. Where possible, we compare repeated projects between the control and treatment groups, but for new projects, we seek to compare with similar projects. To facilitate this comparison, we selected the cyclomatic complexity of the reference solution and the number of mutants generated for it, as metrics of comparison for the "difficulty" of the projects. Cyclomatic complexity is calculated as the number of linearly independent paths through a program and we calculate this value with PMD at the class level [16]. Mutations generated are the number of mutants, or variants of the program, with specifically seeded faults (on lines such as arithmetic or control flow) that are evaluated against the provided test cases to ensure good tests catch these faults. We calculate this value by running mutation testing with PIT using a selected set of mutators [17, 11, 18].

Table 1 shows the project topics and their relative complexities. For example, we compare Fall 2025 Project 1 with Spring 2024 and Fall 2024 Project 1 as they have similar cyclomatic complexity and number of mutants generated by the reference solutions. Project 1 also tends to be an "easier" project and usually only involves one data structure instead of two, unlike most of Projects 2 and 4.

To explore **RQ1**, we selected Cochran's Q test to investigate potential differences in test validation usage rates throughout the semester. For **RQ2**, we run pairwise Mann-Whitney U tests

between similar projects to identify differences in the correctness before and after the intervention as evidenced by reference test pass rates.

Table 1: Project Solution Complexity

Semester	Topic	Condition	Cyclomatic Complexity	Mutations
S24 P1	Skip List	Control	110	174
S25 P1	Skip List	Treatment	110	174
S24 P2	Quad Tree	Control	225	310
S24 P4	Memory Manager	Control	129	180
F24 P4	Memory Manager	Treatment	129	180
F24 P1	Graph	Control	119	186
S25 P4	Graph	Treatment	119	186
F25 P1	Sparse Matrix	Treatment	111	172
F24 P2	Bintree	Control	188	372
F25 P4	Bintree	Treatment	245	620
F25 P2	KD Tree	Treatment	105	182

Evaluation Results

RQ1: Usage Patterns: As the data for the Fall 2025 semester better integrated the validation system into the projects' natural testing, usage of the system was required by the students, whether or not they used the feedback because validation ran on their "default" testing file. However, in the Spring 2025 semester, the system used a separate test file, "ProblemSpecTest.java", instead of the preexisting file students have been trained to write tests in "¡ExampleProjectNameTest¿.java". As this version of the system required students to interact with the system voluntarily, we were able to measure this behavior. This allowed us to analyze how students engaged with the validation system over the course of the semester. We analyzed the usage patterns of 330 students who submitted solutions to the three projects using validation. As these data are longitudinal, consisting of the same students over three projects, we selected Cochran's Q test to identify an overall difference in the validation usage rates. Post-hoc analysis was completed using McNemar's test with a Bonferroni correction for the pairwise comparisons to identify differences in usage across the individual projects.

We hypothesize that students would increasingly adopt the validation system as they became more familiar with its benefits, as the projects increased in difficulty, and as a result of pressure to use it from course staff. A Cochran's Q test confirmed a statistically significant difference in usage rates across the three projects ($\chi^2(2) = 23.36$, $p < 0.001$), indicating a non-random shift in student behavior.

Table 2 shows how many students opted in to using test validation (by writing tests in the designated file) for the three projects in the Spring 2025 semester. We can see here that initial adoption was strong, with 80.0% of students using validation for at least one submission during project 1. Table 3 indicates that while that initial adoption was strong, Project 2 did not see statistically significant changes that we might expect. For Project 4, there was a significant increase in the use of the validation system, from approximately 80.0% use in Projects 1 and 2 to approximately 90.0% use in Project 4.

Table 2: Raw Student Test Validation Usage Change ($N = 330$ Students)

Project	Usage	Usage Rate
P1	264	80.00%
P2	259	78.48%
P4	299	90.61%

Table 3: Student Test Validation Usage ($N = 330$ Students)

Comparison	Usage Rate Change	Difference	p-value
P1 to P2	80.00% $\rightarrow$ 78.48%	-1.52%	0.679
P2 to P4	78.48% $\rightarrow$ 90.61%	+12.13%	< 0.001
P1 to P4	80.00% $\rightarrow$ 90.61%	+10.61%	< 0.001

RQ2: Submission Analysis: To evaluate the impact of test validation on student project correctness, we analyzed 5,349 final submissions for projects over four semesters.

We hypothesize that test validation helped students develop solutions with higher correctness as identified by reference test pass rates. To test this hypothesis, we used pairwise Mann-Whitney U tests between similar projects. To check our assumptions, Shapiro-Wilk tests [19] were conducted to determine the normality of the data (W ranged from 0.390 to 0.804, all $p < 0.001$). Levene's tests [20] revealed that six of the comparisons were heteroscedastic, or had variances that were significantly different, while two were homoscedastic (W ranged from 0.338 to 208.200, with 6 of 8 pairs showing $p < 0.05$).

As a result of the non-normality and heteroscedasticity of the data, we carried out non-parametric pairwise Mann-Whitney U tests [21] with a Bonferroni correction to compare the distributions of reference test pass rates between semesters. We report these results in Table 4.

Table 4: Pairwise Mann-Whitney U Test Results for Reference Tests Comparisons

Comparison	Project	n_1/n_2	Mdn_1 [IQR]	Mdn_2 [IQR]	U	p_{adj}	r
S24 P1 v S25 P1	RectDB	413/494	1.00 [0.94, 1.00]	1.00 [0.94, 1.00]	105,031	1.000	-0.03
F24 P1 v S25 P4	Graph	516/452	0.93 [0.42, 1.00]	1.00 [0.85, 1.00]	84,176	$< .001$	0.28
F24 P2 v F25 P2	Tree	465/455	0.88 [0.61, 1.00]	0.91 [0.77, 1.00]	88,182	$< .001$	0.17
S24 P4 v F24 P4	MemMgr	363/437	0.91 [0.78, 1.00]	1.00 [0.92, 1.00]	49,637	$< .001$	0.37
S24 P2 v F25 P4	Spatial	378/439	1.00 [0.81, 1.00]	0.94 [0.65, 1.00]	90,210	.185	-0.09
F24 P2 v F25 P4	Spatial	465/439	0.88 [0.61, 1.00]	0.94 [0.65, 1.00]	85,336	$< .001$	0.16
S24 P1 v F25 P1	P1	413/468	1.00 [0.94, 1.00]	1.00 [0.96, 1.00]	90,123	.285	0.07
F24 P1 v F25 P1	P1	516/468	0.93 [0.42, 1.00]	1.00 [0.96, 1.00]	72,697	$< .001$	0.40

Note. p-values were adjusted via the Bonferroni correction. r represents the effect size.

The hypothesis was supported in five of the eight comparisons, showing small to medium effect sizes (0.16 - 0.40), indicating improvements in project correctness. Three of the comparisons were not found to be significant, which can likely be attributed to a ceiling effect where the project score medians for both control and treatment groups were already at 100%. One comparison

indicated a significant decrease in performance, notably while reusing the same project. These findings indicate that, while test validation may have some small impact toward improving project correctness, its impact is limited by preexisting high project performance.

Discussion

The small to medium effect sizes observed suggest that while there can be a benefit to using the validation system, there are other factors at play and project or course context plays a role as well. In particular, the Graph project, given in Fall 2024 as Project 1 and Spring 2025 as Project 4, exhibited a moderate impact of test validation on reference test scores. As this was the last project in Spring 2025, students by then had understood the benefits of validation and were better able to take advantage of the system. Related to this, the relative ease of projects such as Project 1 from Spring 2024 and Spring 2025 resulted in a ceiling effect, where the baseline performance on the project was high enough not to see a benefit from validation.

Interviews with teaching assistants (TAs) who interacted with the test validation system indicated that a significant failure of the system was a lack of understanding by the students. Multiple TAs indicated that they had to explain test validation to many of the students who attended their office hours, indicating that we must do a better job of educating students on the system. We expected students would appreciate the opportunity to query the reference implementation much more than they seemed to. The TAs noted that even if students were not actively using validation, if they attended an office hour and had tests failing validation, it gave them a good starting point to begin the interaction. This use case was more prevalent in the Fall 2025 semester, where use of test validation was more natural for students, even if acknowledgment of its results may not have been.

Interviews with students indicated similar findings. One student noted that their previous CS2 course instructed them to write tests for each class in a solution in a test file specifically for that class (e.g., “Hash.java” in “HashTest.java”), creating some confusion as this course attempted to explain the new system while also breaking from their established testing norms. They also mentioned that they did not feel an incentive to use the validation, despite our assumption that students would appreciate access to an “oracle” that could clarify their confusion. Using test validation is a more active type of help-seeking behavior, where the student drives the process for themselves. Attending office hours to seek the answer from a TA is a more passive learning process for the student in many cases. This illustrates the need to consider how to develop student buy-in to develop skills that will help them in their futures, when TAs as oracles no longer exist. While requiring the use of validation may help students develop debugging skills, we want to preserve student agency as much as is reasonable, and increasing requirements to use tools such as test validation can further constrain these projects. Keeping in mind the cognitive load of additional tools is increasingly important as we continue to augment and improve automated feedback systems.

One student noted that for Project 4, they understood the point of test validation and were able to use it successfully to identify misconceptions in their project. That student found the greatest benefit from test validation at the beginning of the project. At the beginning, validation allowed them to ensure that their understanding of the specification was more or less correct to start with.

To this end, validation may be better suited for more complicated projects, especially those with multiple data structures or a complicated internal structure where it is easy for a student to misunderstand some requirement. For example, Project 4 from Fall 2025 was identified as the most difficult project in our dataset by its cyclomatic complexity, and notably by its number of mutations generated, almost twice as many as the next most complex. This project required students to implement a three-dimensional Bintree, a complicated data structure with many possible branches and conditions for testing. Although this project was significantly more difficult than the projects it was compared to, in statistically significant cases, the correctness of this project was higher than that of the control group. Test validation could also increase in value as the difficulty of debugging the projects increases.

An important confounding variable to note across the control and treatment groups in this study is the presence of LLMs and Generative AI tools. For the Spring 2024 and Fall 2024 semesters, AI tools were developing but were not able to complete projects of this scale. By Spring 2025, these tools had developed to a point where students could make significant progress by feeding prompts into an LLM, potentially disrupting the previous system. In Fall 2025, this course made the decision to allow (but not require) LLM use on projects, along with required documentation of use. While we do not believe, for the most part, students were able to pass entire specifications to a tool and receive a working solution, it is impossible to ignore the fact that these tools have increased in sophistication and are able to provide significant pieces of projects. From our interview results, we believe that there is a pedagogical benefit to this system in a course where students may leverage LLM tools to develop their solutions. In particular, as many students in our course took to AI to develop their test cases, having an automated mechanism to inform students when the AI was diverging from the project specification was useful. According to TAs we interviewed, even if the students were not using validation themselves, the TAs were able to quickly spot issues in student test suites that in some cases approached hundreds of tests and would have otherwise overwhelmed human resources.

Finally, a fringe benefit is that test validation helped course staff identify bugs and inconsistencies in the reference implementations. In previous semesters, a bug in a solution might hide for much of a project's life cycle, obfuscated by hidden reference tests. With the test validation system, students can critically analyze the reference implementation's feedback on their test cases, identifying edge cases the reference implementation did not handle, inconsistencies that were not specified, or bugs that slipped through and were not tested adequately.

Threats to Validity

Our evaluation has several potential threats to validity that must be considered when interpreting the results.

Internal Validity: While four of the projects were repeated across the control and treatment groups, there were five projects that were not repeated. Differences in these projects could contribute to differences in reference test scores despite conceptual and structural similarities. To mitigate this threat, we measured proxies for the difficulty of the projects such as cyclomatic complexity and the number of mutants generated. Additionally, starting with Project 2 in Fall 2025, students were told they would be graded on their test validation results, in particular that

they needed a certain number of tests and could not have any validation failures, which will clearly affect the rate of adoption.

External Validity: The data collected for this study were collected from a CS3 Data Structures course at a large, public R1 university. This course, in particular, uses large, multi-week programming projects with complex specifications. The use of this type of assignment may limit the generalizability of the results, as smaller, more concise projects may not exhibit misconceptions similarly. As with any intervention spanning the last few years, this could be impacted by factors such as the ever-increasing availability of LLMs and Generative AI.

Construct Validity: Usage of the test validation system was measured via student use at least once during the project. This use of a binary does not capture the full depth of student use of the system. Additionally, mutation testing metrics and cyclomatic complexity were used as proxies for project difficulty, but neither of these measures was designed for this purpose. Our selection of reference test pass rate as a proxy for correctness is potentially noisy, as we had to exclude one project, Spring 2025 Project 2, where the grading requirement was lowered from 100 points to 75 points (where 50 come from reference tests). It appears that reference test pass rate may also proxy effort or student motivation as well as correctness.

Conclusion Validity: In addition to the quasi-experimental nature of this work, many of the projects studied already had high median correctness values, leading to a ceiling effect that makes it difficult to detect improvements. Despite the use of the Bonferroni correction, multiple pairwise Mann-Whitney U tests created fragmented results that indicate outcomes that may not be uniform, restricting claims about the system's impact.

Despite these threats, the small, consistent observed improvement across several comparable projects suggests that test validation can help improve the correctness of student submissions.

Implications for Practice

The use of this reference test validation system indicated several important considerations for computer science instructors to take into account.

Test validation requires students to consider the requirements of the projects as they develop their solution. This provides instructors with an objective measure of whether students truly understand the specification and what they are supposed to implement. It also allows instructors to provide students with explicit feedback when they misunderstand part of the specification and test a solution that does not match the expectation.

Test validation serves as a more active system for students to seek help from. While students may be interested in seeking more passive forms such as asking LLMs, encouraging them back to active systems allows them to debug their own solutions and serve as a more active participant in their learning. It also can help students bridge the gap between testing and debugging, where a student may trust their test until proven otherwise.

In addition to these benefits for students and instructors, test validation helps TAs diagnose errors in student solutions and provides a streamlined process for helping students in office hours. It can

also reduce the dependence on TAs and course staff as "oracles" by providing an interface for students to answer their own basic specification and format questions.

Finally, while the question of whether or how students should use AI in contexts such as our course context remains unanswered, test validation can help scaffold its use in either case. A common pattern we have identified is students using AI to generate their test cases, and test validation can help mitigate situations where the AI tool loses the context of the assignment and moves in a different direction. In projects where the use of AI tools is forbidden, this system can serve as a check that students actually understand their tests or their code enough to satisfy the system through additional friction when generating solutions and additional snapshots of code to compare.

Conclusions and Future Work

This study presents the results of including a validation system for student test cases in a post-CS2 Data Structures course. We find that students increased their usage of the system over time, possibly as project difficulty increased or as instructions provided by course staff resulted in student understanding of the validation system. While there was not uniform improvement to project correctness across all projects studied, five of the eight comparable project pairs indicated statistically significant improvements with small to moderate effect sizes. These results suggest that test validation feedback can help increase student project correctness under certain circumstances, but the effects are influenced by project complexity and preexisting performance levels.

Further research is needed, particularly on how students' use of this system impacts their help-seeking behavior generally, and specifically with regard to how often they attend office hours with questions about reference test failures. While improving project correctness was a useful outcome of test validation, the main benefit should be how such a system can impact student needs from course staff. Reducing students' reliance on TAs for information about the reference tests can transition student learning to a more active approach and optimizes office hours to help students who truly need assistance. It is important to follow the long-term effects of test validation as it becomes a larger part of projects in this context. We plan to further investigate students' behavior around test validation, analyzing how students interact with the results from validation via IDE event-streams.

Acknowledgments

This work is supported in part by the National Science Foundation under award CCRI-2213790. We thank the anonymous peer reviewers whose feedback improved this paper.

References

[1] Y. Qian and J. D. Lehman, "Using Targeted Feedback to Address Common Student Misconceptions in Introductory Programming: A Data-Driven Approach," *SAGE Open*, vol. 9, p. 2158244019885136, Oct. 2019.

[2] M. Domino, A. Thompson, A. Hicks, A. Jamieson, K. Parajuli, B. Edmison, S. Edwards, and C. Shaffer, "How can we know when we see it? a systematic review of cognitive control skills and behaviors," *ACM Transactions on Computing Education (submitted for review)*, 2024.

[3] S. H. Edwards, "Using software testing to move students from trial-and-error to reflection-in-action," in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, (Norfolk Virginia USA), pp. 26–30, ACM, Mar. 2004.

[4] J. Wrenn and S. Krishnamurthi, "Executable Examples for Programming Problem Comprehension," in *Proceedings of the 2019 ACM Conference on International Computing Education Research*, (Toronto ON Canada), pp. 131–139, ACM, July 2019.

[5] A. M. Shin and A. M. Kazerouni, "A Model of How Students Engineer Test Cases With Feedback," *ACM Transactions on Computing Education*, vol. 24, pp. 1–31, Mar. 2024.

[6] L. Bijlsma, N. Doorn, H. Passier, H. Pootjes, and S. Stuurman, "How do Students Test Software Units?," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pp. 189–198, May 2021.

[7] G. R. Bai, J. Smith, and K. T. Stolee, "How Students Unit Test: Perceptions, Practices, and Pitfalls," in *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, (Virtual Event Germany), pp. 248–254, ACM, June 2021.

[8] L. Porter, C. Taylor, and K. C. Webb, "Leveraging open source principles for flexible concept inventory development," in *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education - ITiCSE '14*, (Uppsala, Sweden), pp. 243–248, ACM Press, 2014.

[9] N. Doorn, T. E. J. Vos, B. Marín, H. Passier, L. Bijlsma, and S. Cacace, "Exploring students' sensemaking of test case design. An initial study," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 1069–1078, Dec. 2021.

[10] M. Aniche, C. Treude, and A. Zaidman, "How Developers Engineer Test Cases: An Observational Study," *IEEE Transactions on Software Engineering*, vol. 48, pp. 4925–4946, Dec. 2022.

[11] R. S. Mansur, C. A. Shaffer, and S. H. Edwards, "Mutating Matters: Analyzing the Influence of Mutation Testing in Programming Courses," in *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1*, (Virtual Event NC USA), pp. 151–157, ACM, Dec. 2024.

[12] S. Prasad, B. Greenman, T. Nelson, and S. Krishnamurthi, "Conceptual Mutation Testing for Student Programming Misconceptions," *The Art, Science, and Engineering of Programming*, vol. 8, p. 7, Oct. 2023.

[13] S. Prasad, B. Greenman, T. Nelson, J. Wrenn, and S. Krishnamurthi, "Making Hay from Wheats: A Classsourcing Method to Identify Misconceptions," in *Proceedings of the 22nd Koli Calling International Conference on Computing Education Research*, (Koli Finland), pp. 1–7, ACM, Nov. 2022.

[14] K. Buffardi and S. Edwards, "Reconsidering Automated Feedback: A Test-Driven Approach," *SIGCSE 2015 - Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pp. 416–420, Feb. 2015.

[15] E. Spertus and P. Jiménez, "Autograding Java Assignments in Gradescope with Jacquard," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE 2024, (New York, NY, USA), p. 1905, Association for Computing Machinery, Mar. 2024.

[16] T. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, pp. 308–320, Dec. 1976.

[17] M. Delahaye and L. du Bousquet, "Selecting a software engineering tool: Lessons learnt from mutation analysis," *Software: Practice and Experience*, vol. 45, no. 7, pp. 875–891, 2015.

[18] A. M. Kazerouni, J. C. Davis, A. Basak, C. A. Shaffer, F. Servant, and S. H. Edwards, "Fast and accurate incremental feedback for students' software tests using selective mutation analysis," *Journal of Systems and Software*, vol. 175, p. 110905, May 2021.

[19] S. S. SHAPIRO and M. B. WILK, "An analysis of variance test for normality (complete samples)†," *Biometrika*, vol. 52, pp. 591–611, Dec. 1965.

[20] M. B. Brown and A. B. and Forsythe, "Robust Tests for the Equality of Variances," *Journal of the American Statistical Association*, vol. 69, pp. 364–367, June 1974.

[21] H. B. Mann and D. R. Whitney, "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other," *The Annals of Mathematical Statistics*, vol. 18, pp. 50–60, Mar. 1947.