Instructor-Written Hints as Automated Test Suite Quality Feedback

James Perretta
Northeastern University
Boston, MA, USA
perretta.j@northeastern.edu

Andrew DeOrio
University of Michigan
Ann Arbor, MI, USA
awdeorio@umich.edu

Arjun Guha
Northeastern University
Boston, MA, USA
a.guha@northeastern.edu

Jonathan Bell
Northeastern University
Boston, MA, USA
j.bell@northeastern.edu

Abstract

Mutation testing measures a test suite's ability to detect bugs by inserting bugs into the code and seeing if the tests behave differently. Mutation testing has recently seen increased adoption in industrial and open-source software but sees limited use in education. Some instructors use manually-constructed mutants to evaluate student tests and provide general automated feedback. Additional tutoring requires more intensive instructor interaction such as in office hours, which requires substantial resources at scale. Prior work suggests that students benefit from frequent, actionable feedback, and our work focuses on the challenge of leveraging automation to give students high-quality feedback when they need it.

We deployed an automated hint system that provides instructor-written hints related to mutants that student-written tests do not detect. We evaluated our hint system in a controlled experiment across four assignments in two introductory programming courses, comprising 4,122 students. We also analyzed student test suite revisions and conducted a mixed-methods analysis of student hint ratings and comments collected by the automated hint system.

We observed a small, statistically significant increase in the mean number of mutants detected by students who received hints (experiment group) compared to those who did not (control group). In 25% of instances where students received a hint, they detected the mutant in a single revision to their test suite. We conclude with recommendations based on our analysis as a starting point for instructors who wish to deploy this type of automated feedback.

CCS Concepts

• **Software and its engineering** → *Software maintenance tools*; • **Applied computing** → *Computer-assisted instruction*;

Keywords

mutation testing, software testing education, automated feedback

SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0531-1/25/02
https://doi.org/10.1145/3641554.3701866

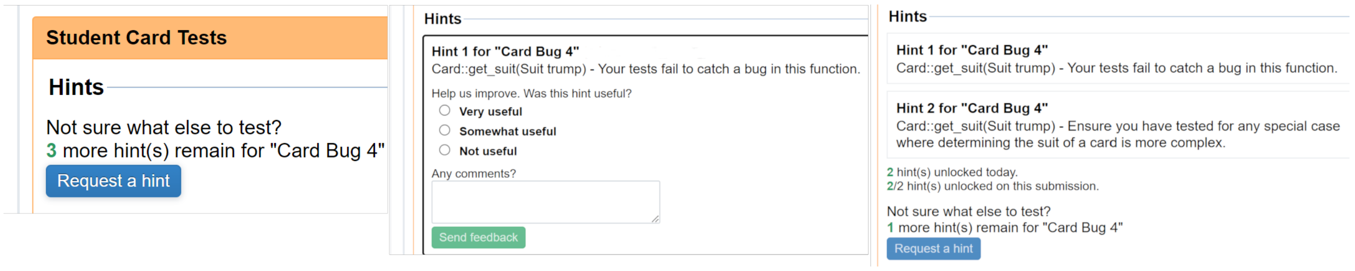
ACM Reference Format:
James Perretta, Andrew DeOrio, Arjun Guha, and Jonathan Bell. 2025. Instructor-Written Hints as Automated Test Suite Quality Feedback. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701866

1 Introduction

Despite a growing body of software testing education research, software testing has long been under-emphasized in computer science curricula [7, 10, 13, 16, 20]. Prior work has explored how to *teach* testing practices, but *evaluating* students' tests and providing feedback, especially as class sizes increase, poses different challenges. There is evidence that students benefit from frequent, actionable feedback [2–5, 10, 18, 19, 21, 24, 25], and leveraging automation to provide feedback to students holds the promise of enhancing student learning at large and small scales. Although there is a large body of research concerned with providing students with automated feedback on the quality of their *implementations* [8, 11, 17, 23], there is still relatively little work examining how to provide high-quality automated feedback to students on the quality of their *test suites*.

The first challenge when evaluating student test suites is choosing how to measure test suite quality. *Code coverage*, which records which portions of the source code are exercised by the test suite, is relatively easy to deploy and interpret its results and is somewhat common in the classroom [10]. However, code coverage is fundamentally limited in its ability to measure bug-finding ability [12]. *Mutation testing* is a way to measure a test suite's ability to detect bugs by making small changes to the source code (each change to the source code is called a *mutant*) and seeing if the tests behave differently, thereby *detecting* the mutant. The number or percentage of mutants that the tests detect is referred to as *mutation score*. Mutation score has been shown to correlate with real bug detection in industrial software [14] and student-written code [22].

The second challenge is providing effective feedback to students. One such practice is to evaluate student test suites against an instructor-written set of mutants and give students limited automated feedback about how many or which mutants their tests detected [21, 26]. Tutoring students on how they can address gaps in their test suites often requires more intensive instructor interaction such as in office hours, which makes this kind of evaluation difficult


Student Card Tests

Hints

Not sure what else to test?
3 more hint(s) remain for "Card Bug 4"

Hints

Hint 1 for "Card Bug 4"
Card::get_suit(Suit trump) - Your tests fail to catch a bug in this function.

Help us improve. Was this hint useful?
Very useful
Somewhat useful
Not useful
Any comments?

Hints

Hint 1 for "Card Bug 4"
Card::get_suit(Suit trump) - Your tests fail to catch a bug in this function.

Hint 2 for "Card Bug 4"
Card::get_suit(Suit trump) - Ensure you have tested for any special case where determining the suit of a card is more complex.

2 hint(s) unlocked today.
2/2 hint(s) unlocked on this submission.

Not sure what else to test?
1 more hint(s) remain for "Card Bug 4"

Figure 1: Screenshots of our hint system. The hints were written by instructors, and students could request an instructor-configured number of hints per submission.

to scale and results in higher-latency feedback for students. Furthermore, a recent study concluded that opaque automated feedback can inadvertently increase demand for office hours [9].

We deployed an automated hint system that lets students request instructor-written hints about mutants their tests did not detect. In our evaluation, we seek to answer the following research questions:

(1) RQ1: Does access to an automated hint system increase student test suite quality?
(2) RQ2: What is the relationship between hints and student test suite revision?
(3) RQ3: What kinds of hints do students perceive as helpful?

2 Methods

We collected data from eight assignments across two terms from four courses at two institutions. Table 1 presents a summary of the assignments. Here we describe the assignment grading and automated feedback, our automated hint system, our datasets, and the controlled experiment we conducted.

2.1 Assignment Grading and Feedback

Every assignment we collected data from shares certain structural elements. Students were required to implement code to conform to a specification and write their own test suites for the implementation. Students could submit their implementation and tests to an automated grading system and receive instructor-configured feedback multiple times per day. Students' implementations were graded against an instructor-written test suite. Students' test suites were graded against a set of instructor-written mutants. Students were awarded points for each mutant their test suite detected.

Students were shown one of two types of automated feedback on their test suites: 1) "Number of mutants" shows how many mutants their tests detected but no other information about what kind of bugs the mutants contain; 2) "Mutant names" shows the instructor-chosen names of the mutants their tests detected and/or did not detect. Instructor-chosen mutant names generally indicate which part of the implementation the mutant affects. Table 1 indicates which assignments used which type of feedback.

Mutant Construction. The mutants were authored by instructors with the goal of representing potential bugs in the full range of implementation behaviors. They did not include adversarial mutants that require obscure test inputs to discover. Prior work by Andrews et al. [1], Kazerouni et al. [15], and Buffardi et al. [6], as

```
int Matrix_width(const Matrix* mat) {
  return mat->~~width~~height;
}
List<T>& List<T>::operator= (const List &l) {
  ~~if (this == &l) return *this;~~
  removeAll();
  copyAll(l);
  return *this;
}
```

Figure 2: Examples of instructor-written mutants. The first is in an accessor for a matrix type. The second is in an overloaded operator for a doubly-linked list.

well as our recent prior work [22], suggests that instructors could use automatically-generated mutants instead of authoring by hand. However, our prior work also found that detection of instructor-written mutants authored in this way is strongly correlated with detection of mutants generated automatically using open source mutation testing tools [22]. Additionally, many of the mutants the instructors in our study authored are syntactically identical to mutants that a mutation testing tool would generate. Figure 2 shows some examples of mutants used.

2.2 Hint System & Hint Development

We implemented a system that lets students request instructor-written hints as additional feedback. We asked instructors to write a sequence of hints for each mutant that would mimic the kind of feedback they would offer to a student who asked a series of questions about why their tests do not detect more mutants. We suggested that the sequence of hints for a particular mutant could increase in specificity (e.g., describing what part of the implementation the mutant affects vs. describing the specific behavior it alters), but we did not set any requirements for how vague or specific the hints should be. We configured the instructor-written hints to be shown through our hint system with instructor-specified limits on how many hints students could request per submission. Table 1 shows the limits that instructors chose per assignment. Figure 1 shows screenshots of the hint system's student interface.

2.3 Datasets

To address our research questions, we required the following:

RQ1. The number of mutants detected by the final revision of each student-written test suite.

Assignment Name	# Students		Control Group Mutant Feedback	# Submissions/Day	# Hints/Submission	
	Fall ’23	Spring ’24			Fall ’23	Spring ’24
CS1 P1	1040	634	Mutant names	4	n/a	2
CS2 P1	777*	746*	Number of mutants	3	n/a	2
CS2 P2	777*	746*	Number of mutants	3	n/a	2
CS2 P3	n/a	756*	Number of mutants	3	n/a	2
CS2 P4	762*	698*	Number of mutants	3	n/a	n/a
PL P1	n/a	72*	Number of mutants	3	n/a	1
PL P2	n/a	67*	Number of mutants	3	n/a	1
SE P1	n/a	253	Mutant names	5	n/a	2

Table 1: Summary of assignments in the study gathered from courses in an intro sequence (CS1, CS2), programming languages (PL), and software engineering (SE). * indicates assignments with optional student partnerships where we report the number of individual students/partnerships who submitted the assignment. Control Group Mutant Feedback indicates the kind of automated feedback that students received on their test suites before the addition of instructor-written hints.

RQ2. Student test suite revision history, computed as follows. First, we extracted the names and source code of each test case per submission. Then, for each adjacent pair of submissions, we compared the names of test cases. A test case name present on the left but not the right was marked as a removed test case. A test case name present on the right but not the left was marked as an added test case. For each test case name present in both submissions, we compared the extracted source code of both test versions. If the source code was different, we marked that as a changed test case.

We also collected each instance where students received a hint and recorded their “outcomes.” That is, we recorded whether the student detected the mutant before requesting another hint (“Mutant Detected”), whether they requested another hint before detecting the mutant (“Hint Requested”), or neither (“Nothing”). We recorded the outcome and the number of revisions until that outcome.

RQ3. Student ratings of hints they received. Our system prompted students to rate hints as “Very Useful”, “Somewhat Useful”, or “Not Useful” and optionally leave comments.

2.4 Controlled Experiment

We conducted a controlled experiment as part of our effort to address RQ1 and RQ2. We compared two terms of each of the following four assignments: CS1 P1, CS2 P1, CS2 P2, and CS2 P4. Our control group is comprised of students in the first term of each assignment (Fall 2023). Our experiment group is comprised of students in the second term of each assignment (Spring 2024).

RQ1 (Student test quality). We compared the distributions of the number of mutants detected by students’ test suites from the control and experiment groups. On assignments CS1 P1, CS2 P1, and CS2 P2 (where the control group did not receive hints but the experiment group did), we look for differences in the mean number of mutants detected as an indication that access to hints helps students write higher quality test suites. On assignments CS2 P1, CS2 P2, and CS2 P4, we look for evidence of a lasting effect on students ability to write high-quality test suites. That is, we examine whether the experiment group wrote higher quality test suites than the control group even after access to hints was removed.

RQ2 (Test suite revision). In addition to our analysis of “hint outcomes” in section 3.2, we perform a comparable analysis to that of RQ1, looking instead at the number of test suite revisions it took for students to reach the final number of mutants they detected.

3 Evaluation

3.1 RQ1: Does access to an automated hint system increase student test suite quality?

Table 2 shows the results of our controlled experiment. We examined whether students who received hints detect more mutants than students who did not. On the “CS1 P1” assignment, we do not see a statistically significant difference between the mean number of mutants detected by the control and experiment groups. On the “CS2 P1” assignment, we see the mean number of mutants detected by the experiment group is higher than that of the control group by 0.36. We find this difference to be significantly significant but with a small effect size (rank biserial correlation coefficient r=0.06). On the “CS2 P2” assignment, we do not see a statistically significant difference between the mean number of mutants detected by the control and experiment groups. We note that students in both groups knew their final score on each resubmission, which potentially limits the differences in mean we see. We may see larger effects in how quickly students reached their solution. On the “CS2 P4” assignment, neither the control nor experiment groups had access to the hint system. This lets us examine whether receiving hints has some short-term lasting effect on students’ ability to write higher-quality test suites. We note that the experiment group also received hints on the “CS2 P3” assignment, but we do not compare the control and experiment groups for this assignment because the mutants used in the Spring 2024 semester (the experiment group) differed from the mutants used in the Fall 2023 semester (the control group). We see that the mean number of mutants detected by the experiment group on the “CS2 P4” assignment is 0.6 higher than that of the control group. We find this difference to be statistically significant but with a small effect size (r=0.068).

3.2 RQ2: What is the relationship between hints and student test suite revision?

We address this question using results from our controlled experiment and with a quantitative analysis of revision history and outcomes (e.g., how many revisions after receiving a hint did students

	Number of mutants detected								Number of revisions							
	CS1 P1		CS2 P1		CS2 P2		CS2 P4		CS1 P1		CS2 P1		CS2 P2		CS2 P4	
	Ctrl	Expr*	Ctrl	Expr*	Ctrl	Expr*	Ctrl	Expr	Ctrl	Expr*	Ctrl	Expr*	Ctrl	Expr*	Ctrl	Expr
N	1015.00	607.00	735.00	693.00	714.00	667.00	721.00	665.00	1015.00	607.00	735.00	693.00	714.00	667.00	721.00	665.00
Mean	7.30	7.23	12.64	13.00	11.67	12.05	13.59	14.19	2.76	2.98	2.76	3.21	3.63	3.86	3.61	3.93
Stdev	1.27	1.42	3.66	3.42	6.50	6.52	4.36	3.67	1.76	1.93	1.86	2.09	2.32	2.47	2.56	2.65
Min	0.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00
Q1	7.00	7.00	12.00	13.00	7.00	7.00	12.00	13.00	1.00	2.00	1.00	2.00	2.00	2.00	2.00	2.00
Median	8.00	8.00	14.00	14.00	14.00	14.00	15.00	15.00	2.00	3.00	2.00	3.00	3.00	4.00	3.00	3.00
Q3	8.00	8.00	15.00	15.00	17.00	17.00	16.00	16.00	4.00	4.00	4.00	4.00	5.00	5.00	5.00	5.00
Max	8.00	8.00	16.00	16.00	20.00	20.00	19.00	19.00	12.00	13.00	13.00	14.00	14.00	23.00	21.00	21.00
U	307039.5		239414.5		227482.0		223368.0		289282.0		222460.5		225205.5		220772.0	
p-value	0.900		0.045		0.149		0.026		0.035		<0.001		0.078		0.010	
r	0.003		0.060		0.045		0.068		0.061		0.127		0.054		0.079	

Table 2: RQ1 & RQ2 Controlled Experiment: Descriptive statistics and ANOVA for number of mutants detected and number of test suite revisions on each assignment. * denotes assignments where the experiment group had access to hints.

detect the underlying mutant?). First, we compare the mean number of test suite revisions made by the control and experiment groups. On the "CS1 P1," "CS2 P1," and "CS2 P4" assignments, we see a statistically significant increase in the mean number of revisions from the control group to the experiment group, with the mean number of revisions made by the experiment group being higher by 0.22, 0.45, and 0.32. We see small effect sizes for each of these results, with rank-biserial correlation coefficients of r=0.061, 0.054, and 0.079, respectively. We do not see a statistically significant difference between the mean number of revisions made by the control and experiment groups on the "CS2 P2" assignment.

Next, we examine three categories of outcomes from receiving individual hints: 1) Mutant Detected, where a student detected the underlying mutant after revising their test suite; 2) Hint Requested, where a student requested another hint for the mutant before detecting it (note that for some assignments, students could request multiple hints per submission, making it possible to request another hint before revising their test suite); 3) Nothing, where the student neither detected the mutant nor requested another hint. In aggregate, 25% of instances where students received a hint resulted in the student detecting the related mutant in a single revision. Table 3 shows a per-assignment breakdown of hint outcomes. On 3/7 assignments, we see a higher percentage of "Mutant Detected" than "Hint Requested" outcomes. On the remaining four assignments, we see that between 31%-42% of hints shown resulted in "Mutant Detected." In over 80% of instances where a student was shown a hint, they either detected the mutant or requested another hint. In the latter outcome, students requested the next hint before revising their tests 65% of the time and after a single revision 27% of the time. Table 3 also shows a breakdown of how many revisions it took students to detect mutants after receiving a hint (but before requesting another hint). We see the highest proportion of single-revision detection for the CS2 assignments, with 85%, 84%, and 77% of hints shown resulting in the mutant being detected after a single revision. We see the lowest proportion of single-revision detection for SE P1, with 42% of hints resulting in single-revision detection, 24% of hints resulting in the mutant being detected in 2 revisions, and 21% resulting in the mutant being detected in 3 or 4 revisions.

3.3 RQ3: What kinds of hints do students perceive as helpful?

We take a multi-faceted approach, augmenting our quantitative results with a qualitative examination of specific hints and student feedback. We focus on the the kind of information given in hints and how useful students rate those hints to be, including what points of frustration are indicated by student comments on hints.

3.3.1 What is the relationship between the kind of information in hints and how students perceive those hints? Table 5 shows the breakdowns of student hint ratings with respect to Hint Number. Hints with a higher Hint Number tend to reveal more information about the underlying mutant. We examine these results together with a qualitative analysis examining why some hints were more or less successful.

Hints Rated "Very Useful". In several assignments, we observe that hints rated as "Very Useful" are more likely to have higher Hint Numbers than their "Somewhat Useful" counterparts. For example, Table 5 shows that 26% of "Very Useful" hint ratings vs. 9% of "Somewhat Useful" hint ratings for the CS2 P2 assignment had Hint Number 2, and 4% of "Very Useful" hint ratings vs. 2% of "Somewhat Useful" hint ratings for the CS2 P3 assignment had Hint Number 3. Below are some examples of hints frequently rated as "Very Useful." These hints tend to reveal specific details about the underlying mutant or provide specific guidance on how to construct a test case that detects the mutant.

- CS2 P2: **Simple Player make_trump() Bug #2 - Ensure you have a test that verifies the player is counting the left bower as a trump. For example, a case where they will only order up if the left bower is (correctly) treated as a trump.** Hint Number: 2. This hint describes an edge case and gives some information on how a test case that detects that mutant might be constructed.
- CS2 P3: **Iterator::operator==() Bug #1 - Consider adding a test case that compares a default-constructed iterator with iterators from a list. The default-constructed iterator should not be equal to any of these iterators, not**

Assignment	# Hints Shown	Mutant Detected					Hint Requested	Nothing
		Total	$NR = 1$	$NR = 2$	$2 < NR \leq 4$	$NR > 4$		
CS1 P1	535	47%	74%	20%	5%	1%	40%	14%
CS2 P1	1138	47%	85%	11%	4%	0%	36%	18%
CS2 P2	1897	31%	84%	12%	4%	1%	51%	18%
CS2 P3	4196	35%	77%	14%	7%	2%	54%	10%
PL P1	552	50%	65%	14%	13%	7%	35%	14%
PL P2	543	42%	57%	21%	13%	9%	45%	13%
SE P1	3149	37%	42%	24%	21%	12%	51%	13%
All		38%					49%	13%

Table 3: Hint outcomes per assignment. The "Mutant Detected" column shows the breakdown of how many revisions it took students to detect the mutant ("NR" means "number of revisions").

	CS1 P1	CS2 P1	CS2 P2	CS2 P3	PL P1	PL P2	SE P1
Total # Hints	535	1138	1897	4196	552	543	3149
# Rated Hints	411 (77%)	984 (86%)	1546 (81%)	3375 (80%)	515 (93%)	495 (91%)	2644 (84%)

Table 4: Total number of hints shown to students and how many of those hints were rated by students.

Assignment		Very Useful					Somewhat Useful					Not Useful				
	Hint #:	0	1	2	3	N	0	1	2	3	N	0	1	2	3	N
CS1 P1		52%	33%	13%	1%	82	45%	41%	13%		135	71%	16%	9%	4%	194
CS2 P1		65%	25%	10%		367	65%	29%	7%		446	44%	50%	6%		171
CS2 P2		31%	41%	26%	2%	469	52%	38%	9%	<1%	733	64%	28%	7%	<1%	344
CS2 P3		39%	28%	29%	4%	1045	45%	38%	15%	2%	1497	48%	38%	12%	2%	833
PL P1		43%	57%			113	63%	37%			288	73%	27%			114
PL P2		26%	74%			118	61%	39%			309	63%	37%			68
SE P1		34%	38%	26%	2%	509	42%	40%	15%	3%	1434	63%	26%	10%	1%	701

Table 5: Breakdown of "Hint Numbers" with respect to each possible hint rating. "Hint Number" is the zero-index of the instructor-defined, per-mutant hint ordering.

even an end() iterator. Hint Number: 3. This hint describes what to set up and what post-conditions to check.

We observe a similar trend in the PL assignments where "Very useful" hint ratings are more common for hints with Hint Number 1 than Hint Number 0 (mutants on these assignments only had 2 hints each). This is despite the fact that PL hints are less specific than the CS2 hints above, e.g., "This bug affects error checking in quote." Students may find this kind of hint more helpful if it addresses a gap in their understanding of what classes of behaviors to test, such as one student who commented, "This is very useful, without it, I wouldn't think of do[ing] error checking for quote."

Hints Rated "Somewhat Useful". "Somewhat Useful" was the most common rating overall, with 48.6% of all hints being rated as such by students. These hints are more likely to have lower Hint Numbers. In Table 5 we see that for every assignment, at least 42% of "Somewhat Useful" hint ratings have Hint Number 0, and at least 80% of "Somewhat Useful" hint ratings have Hint Number 0 or 1. These hints tend to reveal general information about what method or behavior the mutant affects, and we observe similar patterns with this kind of hint on CS2 assignments. Below are some examples of hints frequently rated as "Somewhat Useful".

- PL P1: **This bug affects division procedures.** Since this hint only reveals what part of the program being tested the mutant lies in, it seems more likely to be considered useful by students who only need to be pointed in the right direction of where to consider additional test cases. For example, one student commented on this hint, "Shows a general location to look, so I would say it's sufficient."
- SE P1: **This bug results in an incorrect turn order.** Similar to the above hint, this points students toward the class of behaviors that the mutant affects, here describing the particular post-condition that is affected by the mutant.

Hints Rated "Not Useful". While several factors may make hints more likely to be rated as "Not Useful," students may rate the same hint differently depending on context. Based on comments students submitted with their hint ratings, the two most common points of frustration arise when students: 1) receive redundant information; 2) believe that their tests already cover what a hint describes.

For example, all of the CS1 P1 hints with Hint Number 0 remind the student to call the test functions they wrote: "Check that your test functions are called from within startTests in test.cpp." We see in Table 5 that 71% of "Not Useful" hint ratings on this assignment are for this kind of hint. Comments from students who rated these hints as "Not Useful" tend to state that the student already knew this or had already made sure that they were calling all their test functions. However, in one instance, a student commented on a subsequent hint that they realized that their mistake was that they didn't call one of their test functions after all.

On 5/7 assignments, over 60% of "Not Useful" hint ratings were for hints with Hint Number 0. For assignments other than CS1 P1, hints with Hint Number 0 tend to point out the general area of the program being tested where the mutant lies. On the CS2 and PL assignments, the automated grading system was configured to not reveal this information outside of hints, with mutant names being obfuscated instead of showing the more detailed, instructor-facing mutant name. On the SE P1 assignment, the automated grading system was configured to show students the instructor-facing mutant names, resulting in more frequent comments that hints with Hint Number 0 for this assignment were merely restating information revealed by the mutant names.

Another source of frustration is when students believe they already tested the behavior that a hint refers to. For example, one comment on the PL P1 hint "This bug affects division procedures." states, "I did test zero division but didn't catch bugs. This is just a random guessing game." This frustration can even arise with hints that reveal detailed information about the underlying mutant. For example, one comment on the CS2 P3 hint that we used as an example of a hint frequently rated "Very Useful" states, "I have a test that does exactly this."

4 Discussion

We discuss the implications of our results for software testing educators and reflect on the threats to the validity of our conclusions and the efforts that we took to mitigate those threats.

4.1 Implications for Educators

We observed that, on average, students who had access to hints *detected more mutants* and *made more revisions* to their test suites, even after access to hints was taken away. We also observed evidence that certain kinds of hints can help students *make more productive revisions.* Providing this fast feedback loop may have lasting benefits to students' software testing skills and could potentially contribute to a reduction in demand for office hours, freeing up instructor time to address more complex student questions. Our results also provide a starting point in developing best practices for authoring software testing hints.

Avoid redundant information between hints and other sources. We observed a common point of frustration were hints that gave information that students already knew, either from other hints they had received or from other parts of the automated feedback they receive (Section 3.3.1). We recommend not using hints to convey general reminders, and instead give them in some other way.

Receiving multiple hints per submission may not be meaningfully different from receiving one hint with more information. In section 3.2 we observed that in 65% of instances where students requested another hint, they did so before revising their test suite. While allowing multiple hints per submission may be beneficial to some students, we observed student frustration when they were able to request two hints on the same submission but felt that the information in one of those hints was redundant.

"Too much" information may be OK. A common concern in our discussions with instructors was that "giving away" too much information could inhibit student learning. However, our controlled experiment evaluation in Section 3.1 shows that students in the experiment group who had access to the most revealing hints in our study (the CS2 assignments) still detected more mutants on average than students in the control group even after access to hints was taken away. We also observed in Section 3.3.1 that students were more likely to consider very-revealing hints as being very useful, suggesting that this is the kind of feedback students needed.

4.2 Threats to validity

Construct: Are we asking the right questions? Our research questions were prompted by our experience providing tutoring to students when using mutation testing to evaluate student test suites. We posed our research questions before examining our dataset.

Internal: Do our methods and datasets affect the accuracy of our results? Our controlled experiment was conducted using moderately complex programming assignments across multiple terms. There may be unknown confounds arising from differences between students in each term. The course materials (e.g., lecture content, assignments) were synchronized between terms and across instructors within terms, limiting differences in course content.

We used the number of revisions as a measurement of how quickly students improved their test suites. While we could not control for student activities between revisions (e.g., going to office hours), we also examined student hint ratings to provide a fuller picture of how helpful the hints were.

External: Would our results generalize? Our evaluation uses eight programming assignments, and they may not be representative of every kind of programming assignment. However, our assignments were drawn from four different courses across two institutions, comprising 4,122 students.

5 Conclusion

We investigated the impact of an automated hint system using instructor-written hints on students' ability to write higher-quality test suites. Our results show a small, statistically significant increase in mutant detection by students who received hints vs. students who did not. Additionally, we saw that 25% of instances where students received a hint resulted in them detecting the underlying mutant in a single revision, suggesting that certain kinds of hints provided students with the guidance they needed when they needed it. Our mixed-methods analysis of instructor written hints provides a starting point for developing hint-writing best practices. Future research in this area should consider additional kinds of assignments and other strategies for authoring hints.

Acknowledgments

We thank Amir Kamil, James Juett, and William Arthur for using our tool and writing hints, and we thank the students who submitted feedback on those hints.

References

[1] J. H. Andrews, L. C. Briand, and Y. Labiche. 2005. Is mutation an appropriate tool for testing experiments?. In *Proceedings of the 27th International Conference on Software Engineering* (St. Louis, MO, USA) *(ICSE '05)*. Association for Computing Machinery, New York, NY, USA, 402–411. https://doi.org/10.1145/1062455.1062530

[2] Gina R. Bai, Kai Presler-Marshall, Thomas W. Price, and Kathryn T. Stolee. 2022. Check It Off: Exploring the Impact of a Checklist Intervention on the Quality of Student-authored Unit Tests. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1* (Dublin, Ireland) *(ITiCSE '22)*. Association for Computing Machinery, New York, NY, USA, 276–282. https://doi.org/10.1145/3502718.3524799

[3] Gina R. Bai, Justin Smith, and Kathryn T. Stolee. 2021. How Students Unit Test: Perceptions, Practices, and Pitfalls. In *Proceedings of the 2021 Conference on Innovation and Technology in Computer Science Education (ITiCSE '21)*.

[4] Gina R. Bai, Sandeep Sthapit, Sarah Heckman, Thomas W. Price, and Kathryn T. Stolee. 2023. An Experience Report on Introducing Explicit Strategies into Testing Checklists for Advanced Beginners. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 194–200. https://doi.org/10.1145/3587102.3588781

[5] Patrick Bassner, Eduard Frankford, and Stephan Krusche. 2024. Iris: An AI-Driven Virtual Tutor for Computer Science Education *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 394–400. https://doi.org/10.1145/3649217.3653543

[6] Kevin Buffardi, Pedro Valdivia, and Destiny Rogers. 2019. Measuring Unit Test Accuracy. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 578–584. https://doi.org/10.1145/3287324.3287351

[7] David Carrington. 1997. Teaching software testing. In *Proceedings of the Australasian conference on computer science*. 59–64.

[8] Brenda Cheang, Andy Kurnia, Andrew Lim, and Wee-Chong Oon. 2003. On automated grading of programming assignments in an academic institution. *Computers & Education* 41, 2 (2003), 121–131. https://doi.org/10.1016/S0360-1315(03)00030-7

[9] Andrew Deorio and Christina Keefer. 2022. When is Automated Feedback a Barrier to Timely Feedback? https://doi.org/10.18260/1-2--40405

[10] Stephen H. Edwards. 2003. Improving Student Performance by Evaluating How Well Students Test Their Own Programs. *J. Educ. Resour. Comput.* 3, 3 (sep 2003), 1–es. https://doi.org/10.1145/1029994.1029995

[11] Aliya Hameer and Brigitte Pientka. 2019. Teaching the art of functional programming using automated grading (experience report). *Proc. ACM Program. Lang.* 3, ICFP, Article 115 (jul 2019), 15 pages. https://doi.org/10.1145/3341719

[12] Laura Inozemtseva and Reid Holmes. 2014. Coverage is Not Strongly Correlated with Test Suite Effectiveness. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 435–445. https://doi.org/10.1145/2568225.2568271

[13] Edward L. Jones and Christy L. Chatmon. 2001. A perspective on teaching software testing. *Journal of Computing Sciences in Colleges* 16 (2001), 92–100.

[14] 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 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 654–665. https://doi.org/10.1145/2635868.2635929

[15] Ayaan M. Kazerouni, James C. Davis, Arinjoy Basak, Clifford A. Shaffer, Francisco Servant, and Stephen H. Edwards. 2021. Fast and accurate incremental feedback for students' software tests using selective mutation analysis. *Journal of Systems and Software* 175 (2021), 110905. https://doi.org/10.1016/j.jss.2021.110905

[16] Otávio Augusto Lazzarini Lemos, Fábio Fagundes Silveira, Fabiano Cutigi Ferrari, and Alessandro Garcia. 2018. The impact of Software Testing education on code reliability: An empirical assessment. *Journal of Systems and Software* 137 (2018), 497–511. https://doi.org/10.1016/j.jss.2017.02.042

[17] Haden Hooyeon Lee. 2021. Effectiveness of Real-time Feedback and Instructive Hints in Graduate CS Courses via Automated Grading System. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 101–107. https://doi.org/10.1145/3408877.3432463

[18] Marcus Messer, Neil Brown, Michael Kölling, and Miaojing Shi. 2023. Automated Grading and Feedback Tools for Programming Education: A Systematic Review. https://doi.org/10.1145/3636515

[19] Helen J Parkin, Stuart Hepplestone, Graham Holden, Brian Irwin, and Louise Thorpe. 2012. A role for technology in enhancing students' engagement with feedback. *Assessment & Evaluation in Higher Education* 37, 8 (2012), 963–973. https://doi.org/10.1080/02602938.2011.592934 arXiv:https://doi.org/10.1080/02602938.2011.592934

[20] Leo Natan Paschoal, Myke M. Oliveira, Silvana M. Melo, Ellen F. Barbosa, and Simone R. S. Souza. 2020. Evaluating the impact of Software Testing Education through the Flipped Classroom Model in deriving Test Requirements. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering* (<conf-loc>, <city>Natal</city>, <country>Brazil</country>, </conf-loc>) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 570–579. https://doi.org/10.1145/3422392.3422489

[21] James Perretta and Andrew DeOrio. 2018. Teaching Software Testing with Automated Feedback. In *2018 ASEE Annual Conference & Exposition*. ASEE Conferences, Salt Lake City, Utah. https://peer.asee.org/31062.

[22] James Perretta, Andrew DeOrio, Arjun Guha, and Jonathan Bell. 2022. On the use of mutation analysis for evaluating student test suite quality. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (<conf-loc>, <city>Virtual</city>, <country>South Korea</country>, </conf-loc>) *(ISSTA 2022)*. Association for Computing Machinery, New York, NY, USA, 263–275. https://doi.org/10.1145/3533767.3534217

[23] Mark Sherman, Sarita Bassil, Derrell Lipman, Nat Tuck, and Fred Martin. 2013. Impact of auto-grading on an introductory computing course. *J. Comput. Sci. Coll.* 28, 6 (jun 2013), 69–75.

[24] Valerie J. Shute. 2008. Focus on Formative Feedback. *Review of Educational Research* 78, 1 (2008), 153–189. https://doi.org/10.3102/0034654307313795 arXiv:https://doi.org/10.3102/0034654307313795

[25] Rebecca Smith, Terry Tang, Joe Warren, and Scott Rixner. 2017. An Automated System for Interactively Learning Software Testing. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy) *(ITiCSE '17)*. Association for Computing Machinery, New York, NY, USA, 98–103. https://doi.org/10.1145/3059009.3059022

[26] John Wrenn, Shriram Krishnamurthi, and Kathi Fisler. 2018. Who Tests the Testers?. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. Association for Computing Machinery, New York, NY, USA, 51–59. https://doi.org/10.1145/3230977.3230999