A Comparison of Inquiry-Based Conceptual Feedback vs. Traditional Detailed Feedback Mechanisms in Software Testing Education: An Empirical Investigation

Lucas Cordova
Western Oregon University
Monmouth, OR, USA
cordoval@wou.edu

Jeffrey Carver, Noah Gershmel
University of Alabama
Tuscaloosa, AL, USA
carver@cs.ua.edu
nggershmel@crimson.ua.edu

Gursimran Walia
Georgia Southern University
Statesboro, GA, USA
gwalia@georgiasouthern.edu

ABSTRACT

The feedback provided by current testing education tools about the deficiencies in a student's test suite either mimics industry code coverage tools or lists specific instructor test cases that are missing from the student's test suite. While useful in some sense, these types of feedback are akin to revealing the solution to the problem, which can inadvertently encourage students to pursue a trial-and-error approach to testing, rather than using a more systematic approach that encourages learning. In addition to not teaching students why their test suite is inadequate, this type of feedback may motivate students to become dependent on the feedback rather than thinking for themselves. To address this deficiency, there is an opportunity to investigate alternative feedback mechanisms that include a positive reinforcement of testing concepts. We argue that using an inquiry-based learning approach is better than simply providing the answers. To facilitate this type of learning, we present Testing Tutor, a web-based assignment submission platform that supports different levels of testing pedagogy via a customizable feedback engine. We evaluated the impact of the different types of feedback through an empirical study in two sophomore-level courses. We use Testing Tutor to provide students with different types of feedback, either traditional detailed code coverage feedback or inquiry-based learning conceptual feedback, and compare the effects. The results show that students that receive conceptual feedback had higher code coverage (by different measures), fewer redundant test cases, and higher programming grades than the students who receive traditional code coverage feedback.

CCS CONCEPTS

• **Software and its engineering → Software creation and management**; **Software verification and validation**;

KEYWORDS

Testing, Education, Pedagogy, Tools

SIGCSE '21, March 13–20, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8062-1/21/03…$15.00
https://doi.org/10.1145/3408877.3432417

ACM Reference Format:
Lucas Cordova, Jeffrey Carver, Noah Gershmel, and Gursimran Walia. 2021. A Comparison of Inquiry-Based Conceptual Feedback vs. Traditional Detailed Feedback Mechanisms in Software Testing Education: An Empirical Investigation . In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432417

1 INTRODUCTION

In programming courses, students often write and debug code by trial-and-error, running it on sample input (often provided by an instructor), or just using the compiler (i.e. believing that if it compiles, it must be correct) [8]. Students' lack of testing knowledge and ability mistakenly leads them to believe that they can determine the correctness of their code from a small number of test cases. They do not foresee their programs failing other test cases that may be used by their instructor and thus do not understand why they receive a low grade. Moreover, after their formal education, students are not properly equipped to enter the workplace because many graduate with a knowledge gap about software testing [2, 9, 14–17, 26]

Researchers have developed pedagogical tools [4, 6, 23] to address the shortcomings in testing education. These tools (e.g., Marmoset, WebCAT) each provide students with different types of information about their code and tests including information like: whether the instructor's tests have passed, code coverage, and even pointing out exactly which portion of code is problematic. While these tools do provide useful information, one of the main drawbacks is that these tools tend to provide the 'answers' (usually after the students meet some criteria). For example, a tool might tell a student exactly which of the instructor's test cases are missing from his or her test suite. In addition, code coverage tools indicate exactly which portions of code are not fully tested. While this information may be useful for improving the current test suite, beginning students lack the metacognitive ability to use that information to identify their knowledge gaps about fundamental testing concepts and determine why their test suite was inadequate. Tools that provide this type of automated feedback may actually discourage students from thinking on their own and instead encourage them to rely upon the automated feedback [3].

We approach this limitation by focusing on an inquiry-based learning (IBL) approach [20]. IBL is based on the Cognitive Constructivist learning theory [25] that students are more engaged when they actively construct and validate their own knowledge

(through experience) rather than simply receiving the answers. IBL has improved student learning in many disciplines including: Social Science [12], Geography [24], Psychology [18], Medicine [10], Physics [1], Meteorology [27], Chemistry [29], and Forestry [28]. Meta-analyses of IBL experiments showed that, compared with a traditional classroom approach, IBL results in an improved academic achievement, deeper understanding of content, critical thinking skills, motivation, engagement, and creativity [21, 22]. Previous work about IBL in CS (when used to teach the LOGO programming language) has shown that learning should emphasize discovery and that there needs to be some tutor facilitation to guide the learning process rather than leaving it open-ended [11, 19].

To implement an IBL-based approach for improving software testing education, we built **Testing Tutor**, a web-based assignment submission platform that supports testing pedagogy via a customizable feedback engine that can be integrated into any level of the CS curriculum. Testing Tutor's innovative pedagogical contribution is in the type of feedback it provides. Rather than providing a student with the 'answer' (e.g. exactly which tests are missing) or the level of test coverage, Testing Tutor provides students with *conceptual feedback*. *Conceptual feedback* informs the student which underlying fundamental testing concepts their test suites does not adequately cover and provides suggestions for the student to initiate their own learning process about those concepts. This type of feedback will allow the student to determine on his or her own how to improve their test suite, rather than being told by the system exactly what is missing. Examples of *conceptual* feedback for a CS2-level assignment are "the test suite has not fully tested all boundary conditions" and "the test suite misses part of a compound Boolean expression" along with resources provided (examples, videos).

The primary goal of this paper is **to assess whether *conceptual* feedback helps students produce better, more concise and comprehensive test suites.**

2 PREVIOUS APPROACHES

Some educators have developed approaches to improve software testing pedagogy. Testing Tutor builds upon the ideas and shortcomings in these existing approaches.

Collofello and Vehathiri [4] developed a testing simulator that executes student test cases against built-in buggy programs and displays missing test cases. The tool reports the following metrics: 1) *test completeness* – a measure of input coverage, 2) *flow coverage* – a measure of statement/path coverage, 3) *correctness* – a measure of student test outputs correspondence to expected test outputs, and 4) a *fault detection metric* – a measure of fault detection effectiveness. At the conclusion of the testing exercise, the tool gives the students the 'answers' (e.g. the correct set of tests). While this approach does encourage students to improve the test suite for their current project, it does not provide feedback about why the test suite is incomplete or help students carry the knowledge over to future assignments. Testing Tutor borrows the idea of learning how to test by testing someone else's code. But different from this approach, which allows students to succeed through trial-and-error, Testing Tutor provides *conceptual* feedback about why the test suite is incomplete and provides support for learning fundamental testing concepts that should carry over to future assignments.

Marmoset [23] and WebCAT [6–8] use test coverage to provide students with automated feedback on their code. Marmoset provides public tests (i.e. visible to students) and private tests (i.e. not visible to students). When students' code passes the public tests, Marmoset executes the private tests and informs the student of how many private tests failed along with the names of two of those failed tests. Similarly, WebCAT uses a concept of public and private tests to perform automated grading and provides students with feedback during the development process. WebCAT provides students with detailed feedback on failed tests and can provide code annotations with suggestions for improvement. A major drawback to these tools is the extra grading of tests cases required of the instructors. Testing Tutor borrows the idea of encouraging students to write better test cases but does not add additional work for instructors to grade test cases. In addition, rather than providing feedback about specific test cases, Testing Tutor provides guidance on how students can improve their own tests, which should encourage students to better learn the underlying testing concepts.

ProgTest [5] uses two sets of code and test cases, one provided by the student and one provided by the tool or the instructor. ProgTest runs the student's tests against both their own code and the code provided by the tool and runs the tests provided by the tool against the student's code. By comparing the coverage between the student's code and the instructor's code, the student can continuously improve their code and tests. This approach produces higher quality outcomes in terms of semantic correctness compared with developing tests only after developing a complete code solution [13]. Testing Tutor expands upon the type of feedback provided by ProgTest to include *conceptual* information about the tests that are missing.

3 TESTING TUTOR

Testing Tutor is a web-based tool that supports testing pedagogy by helping students learn to develop higher-quality test suites and become more effective testers. Testing Tutor uses a reference implementation and its corresponding test suite to identify which tests are missing from a student's test suite. Each missing test has one or more fundamental testing concept (e.g. testing boundary conditions or testing for data integrity) attributed to it. Based on the missing tests, Testing Tutor helps the students understand which fundamental testing concept knowledge they lack so they can improve their own test suite. Testing Tutor can be used in either *Learning Mode* or *Development Mode*. In Learning Mode, Testing Tutor teaches a student how to develop a complete test suite for a reference implementation of a program. In Development Mode, Testing Tutor helps a student completely test their own newly written code. Learning Mode can be used by itself as a testing exercise or to prepare students to write their own code for later use with Development Mode. If instructors use Learning Mode in a stand-alone fashion (i.e. students will not later implement the assignment), then instructors can decide whether students see the reference code (white-box testing) or only see the specification (black-box testing). If instructors use Learning Mode to prepare students for the Development Mode, then the students will not see the reference code (black-box testing).

Testing Tutor is different from existing software testing education pedagogy because of its' customizable feedback mechanism. Testing Tutor allows an instructor to tailor the level and type of

feedback provided to the students. By annotating the test cases in the reference implementation and configuring the type of feedback Testing Tutor provides, an instructor can choose which learning concepts she or he wants to be the focus. Currently, Testing Tutor supports three types of feedback mechanisms, as discussed in this paper (*detailed* feedback, *conceptual* feedback, and no feedback). This design creates an opportunity for instructors and researchers to investigate which feedback mechanisms best promote learning and improvement while teaching software testing concepts.

The features of Testing Tutor include:

- *Web-based interface* - access to the tool via any web browser;
- *Authorization and authentication management* - supports institution hierarchies for courses, students, faculty, administrators, and assignments;
- *Multi-institution support* - each institution can configure Testing Tutor so that users, assignments, and reports remain separate from other institutions;
- *Assignment repository* - assignments can be private (to the instructor), shared only within an institution, or public;
- *Tailored feedback* - instructors select the type of feedback: no feedback, *detailed* feedback, or *conceptual* feedback;
- *Course management* - generate reports for courses, instructors, and individual students, and analysis for assignments or groups of assignments; and
- *Plug-in platform architecture* - facilitates the addition of more programming languages and is built to scale.

4 EXPERIMENT

To investigate the impact of *conceptual* feedback compared with traditional *detailed* code coverage feedback using Testing Tutor's *Learning Mode*, we pose the following high-level research questions:

RQ1: *How do different types of feedback (conceptual, detailed, none) affect the quality of student test suites?*

Then, to specifically evaluate the usefulness of Testing Tutor, we pose a second research question:

RQ2: *What are the students' perception of the usefulness of Testing Tutor in terms of its usability and the feedback provided?*

To answer these research questions, we conducted a series of two quasi-experiments (an initial study followed by a replication). The remainder of this section provides details on these studies.

4.1 Participating Subjects and Artifacts

We performed the studies in a sophomore-level software testing course at Oregon Institute of Technology in the Spring and Summer 2019 semesters. We chose this course because it has the goal of helping students produce and improve the quality of code. Prior to this course, students completed CS1, CS2 and CS3 (Data Structures).

For each study (the original and the replication), we split the students into two groups based on their course section. The students in Group A received the traditional *detailed* coverage feedback. The students in Group B received the *conceptual* feedback. Table 1 illustrates the number of participants in each group.

Over the course of the study, the students in each group received the same five assignments, all written in Java 1.11. We instructed the students to produce the most comprehensive, yet smallest, test suite possible for each assignment. The five assignments were:

Table 1: Study group compositions

Study	Group A Participants	Group B Participants
Spring 2019	15	16
Summer 2019	13	15

- *Assignment 1* - An I/O program (a calendar program taking a date as input and returning the date of the day before, the day after, one week before, or one week ahead).
- *Assignment 2* - A state-based data structure (a queue) supporting all queue operations and exception-handling.
- *Assignment 3* - An object-oriented calculator containing multiple interfaces and inheritance.
- *Assignment 4* - A comma-separated value (CSV) parser built using the Visitor design pattern.
- *Assignment 5* - A banking application built using the Observer pattern for support of multiple clients.

4.2 Independent Variable – *Type of Feedback*

To measure the impact the *type of feedback* had students' testing performance, we defined the following levels for the independent variable, and configured Testing Tutor accordingly:

- *Treatment A* - Traditional *detailed* feedback similar to code coverage output from tools like JaCoCo and CodeCover.
- *Treatment B - Conceptual* feedback which provides the student with the testing concepts that are not adequately tested from the perspective of the instructor and includes resources to review (textual and video).

Figure 1 provides an example of the *detailed* coverage type of feedback (Treatment A). Figure 2 provides an annotated example of *conceptual* feedback (Treatment B).

4.3 Study Design

We received IRB approval for the studies. We employed the same seven phases in both instances of the course. When using Testing Tutor, we instructed the students to create the smallest, yet most complete test suite possible. We allowed the students to submit their code and tests as often as they would like to receive feedback.

- *Phase 1 - Training* : Training on using Testing Tutor.
- *Phase 2 - Pre-test* : Students completed Assignment 1 using Testing Tutor, configured to provide no feedback (baseline).
- *Phases 3-5 - Data Collection* : Students completed assignments 2-4 using Testing Tutor. Students in Group A received the *detailed* feedback (Treatment A). Students in Group B received the *conceptual* feedback (Treatment B).
- *Session 6 - Post-test* : Students completed assignment 5 using Testing Tutor, configured to provide no feedback. This data helps us understand the students' retention of information learned from the feedback provided by Treatments A and B.
- *Session 7 - Survey* : Students complete a survey about their experience using Testing Tutor and the likelihood they would use it in future courses.

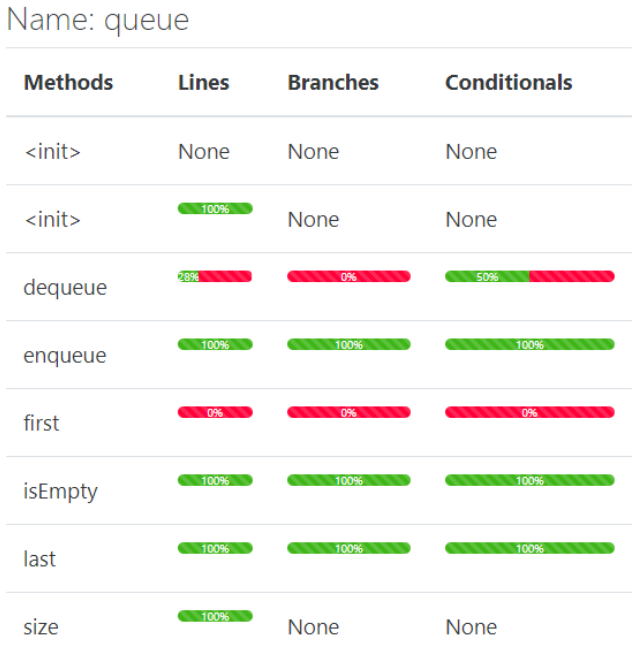

Figure 1: *Detailed* feedback example

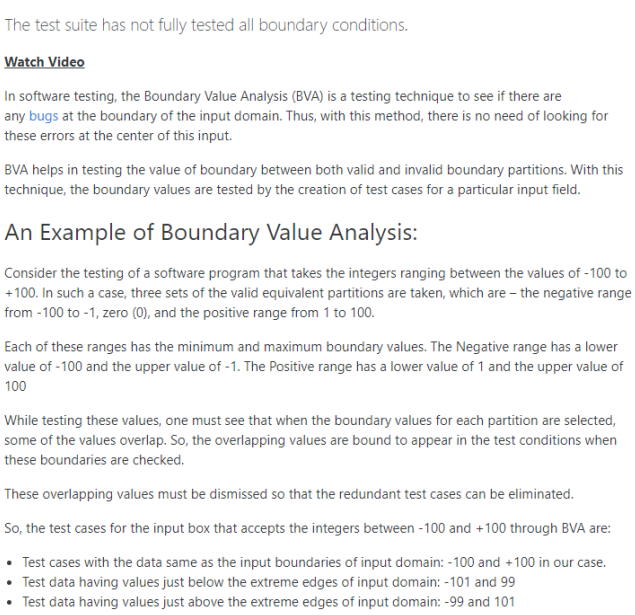

Figure 2: *Conceptual* feedback example

4.4 Dependent Variables

To help answer our research questions, we gathered data to support multiple dependent variables. First, for each assignment submission, Testing Tutor collected the following data:

- *Line coverage* - The percentage of line coverage obtained.
- *Branch coverage* - The percentage of branch coverage obtained.
- *Conditional coverage* - The percentage of conditional coverage obtained.
- *Redundant tests* - The number of tests in the test suite that are redundant (i.e. test code that is tested by another test).

Second, for each assignment we gathered the following data outside of Testing Tutor:

- *Assignment grade* - The instructor assigns a grade for the quality of the test suite based on a rubric[1] that includes code coverage achieved, the number of redundant tests, and a visual inspection of test quality.

Lastly, we collected the following data at the end of the study:

- *Perception of student understanding of the feedback* - An end-of-study optional and anonymous survey gathered the students' perception of the feedback provided by Testing Tutor as well as the usability of Testing Tutor.

5 RESULTS

We organize this section around the study phases (Section 4). We do not have any *a priori* reason to believe that the students in the two semesters are significantly different from each other. Therefore, in the analyses that follow, we combine the data from both semesters. In our ANOVA tests, we include the semester as one of the factors. For the sake of space, we do not include all the details of the ANOVA results in the paper. These are available in the online appendix[2].

5.1 Pre-test

First, we analyzed the performance of the students on the pre-test (Assignment 1) to serve as a baseline and ensure no systematic differences between the groups. Table 2 provides an overview of the data. The results of the 2-way ANOVA tests for each of the five dependent variables collected on Assignment 1 shows:

- No significant differences between the groups for any of the dependent variables
- No main or interaction effects from the semester

Therefore, we can conclude based on the pre-test, that there was no built-in bias in the grouping.

Table 2: Pre-test Results

Dependent Variable	Treatment A (Detailed)	Treatment B (Conceptual)
Line Coverage	35%	35.7%
Branch Coverage	35.3%	34.9%
Conditional Coverage	35.1%	36.6%
Redundant Tests	4.86	4.90
Assignment Grade	57.95%	58.42%

[1] https://github.com/TestingTutor/Data/blob/master/SIGCSE21/Rubric.pdf
[2] https://github.com/TestingTutor/Data/tree/master/SIGCSE21

5.2 Comparison of Approaches

Next, we analyzed whether there were any differences in the dependent variables between the two groups for Assignments 2-4. In this case, the ANOVA tests were slightly more complicated. Because we had no reason to believe the specific assignments would impact the dependent variables, we analyzed Assignments 2-4 together. As a result, we added a factor to the ANOVA. For these five ANOVAs (one for each dependent variable), we have the following factors:

- Treatment (Detailed/Conceptual)
- Assignment (2/3/4)
- Semester (Spring/Summer)

Table 3 overviews the results which show:.

- Significant differences ($p < .001$) for all dependent variables;
- One case (Branch Coverage) in which the Assignment factor had showed a main effect;
- One case (Conditional Coverage) in which the Semester showed a main effect; and
- Two cases (Branch Coverage and Grade) where the Assignment showed an interaction with the Treatment

Overall, the results show students who received the *conceptual* feedback performed significantly better than those who received the *detailed* feedback. At this point, we do not have a good explanation for few cases where other factors showed main or interaction effects, but will continue to explore these in future studies.

Table 3: Main Results

Dependent Variable	Treatment A (Detailed)	Treatment B (Conceptual)
Line Coverage	43.4%	55.1%
Branch Coverage	43.1%	52.7%
Conditional Coverage	45.4%	57.5%
Redundant Tests	4.86	3.33
Assignment Grade	60.37%	68.27%

*[all differences significant $p < .05$]

5.3 Post-test

The results of the post-test (Assignment 5) provide insight into whether the type of feedback received on Assignments 2-4 affected the students' ability to test when that feedback was removed. In Assignment 5, the students received no Testing Tutor feedback. In this case, we ran five 2-way ANOVAs, similar to the pre-test. The results of these tests (summarized in Table 4) showed:

- Significant differences for all five dependent variables; and
- No main or interaction effects from the Semester

Therefore, we can conclude that the type of feedback received did have an effect on learning. The students who received the *conceptual* feedback were able to carry over what they learned more effectively than those received the *detailed* feedback.

5.4 Survey

The end-of-study optional survey contained nine questions each using a 7-point rating scale and three open-ended questions to gather information about usability. To examine whether there was any significant difference between the students in the two groups, we conducted a t-test for each question. Table 5 shows the averages for each group across both studies. In all cases, those receiving *conceptual* feedback viewed Testing Tutor significantly more favorably. Due to space, detailed results are available in the online appendix.

Table 4: Post-test Results

Dependent Variable	Treatment A (Detailed)	Treatment B (Conceptual)
Line Coverage	37.9%	68.8%
Branch Coverage	38.6%	69.4%
Conditional Coverage	44.8%	72.6%
Redundant Tests	4.29	2.29
Assignment Grade	60.31%	78.95%

*[all differences significant $p < .05$]

6 DISCUSSION

The objective of these experiments were to compare the effects of inquiry-based *conceptual* feedback with those of more traditional feedback mechanisms for software testing education. We now discuss insights and possible implications for software testing education as well as the limitations to these studies.

6.1 Answers to Research Questions

To summarize the results in the previous section, we provide an answer for each research question.

RQ1 - Effects of Feedback: Perhaps of greatest practical significance, our results show that students, who began on an equal footing relative to testing knowledge and skill (based on our pre-test), achieved significantly different levels of code coverage, test redundancies, and programming grades based on the type of feedback they received. On average, students who received *conceptual* feedback had higher code coverage (line, branch, and conditions), fewer redundant test cases, and higher programming grades compared with the students who received *detailed* feedback. This increased performance occurred both while the students received the feedback (during Assignments 2-4) and once the feedback was removed (during Assignment 5).

The fact that the effect carried over to Assignment 5 indicates students who received *conceptual* feedback were able to learn how to be better testers, which resulted in their better performance on Assignment 5. Therefore, we can conclude that student obtained more long-term benefits from the *conceptual* feedback than from the *detailed* feedback.

RQ2 - Student Perceptions: The end-of-study survey results illustrated the students' preference for *conceptual* feedback. Students who received *conceptual* feedback indicated that Testing Tutor helped them meet the objectives of the assignments (achieving higher code coverage and reducing redundant tests) in a more productive and effective way than students who received *detailed*

Table 5: Survey Results (all questions on a 7-point scale)

	Question Text	Treatment A (Detailed)	Treatment B (Conceptual)
1	The information that Testing Tutor provided helped me discover deficiencies in the code coverage.	3.57	5.82
2	The information Testing Tutor provided helped me discover redundant tests.	4.25	5.21
3	The information Testing Tutor provided regarding code coverage deficiencies made a lasting impression on how I approach software testing in the future.	3.75	5.52
4	The information Testing Tutor provided regarding redundant tests made a lasting impression on how I approach software testing in the future.	3.18	4.85
5	Testing Tutor helped me become more EFFECTIVE at testing (achieving higher code coverage and reducing redundant tests)	3.43	5.61
6	Testing Tutor helped me become more PRODUCTIVE at testing (achieving higher code coverage and reducing redundant tests during the amount of time spent).	3.82	5.97
7	Testing Tutor is easy to use.	3.89	5.79
8	I learned to use Testing Tutor quickly.	5.14	6.21
9	I would recommend Testing Tutor to someone learning software testing.	3.40	6.21

*[all differences significant $p < .05$]

feedback. The survey results also indicated that *conceptual* feedback had an effect on the students' perception of Testing Tutor's usability, ease of use, and whether they would recommend Testing Tutor to someone learning software testing. All of these results were statistically significant in favor of the *conceptual* feedback.

6.2 Threats to Validity

Internal validity: Sample size is the primary threat to internal validity. The sample sizes were constrained by the size of the class sections at the university during the time the studies were conducted. Although our pre-test showed no built-in bias in the grouping, we used ANOVAs and t-tests, which assume a normal distribution.

Construct validity: Our definition of a high-quality test suite is the primary threat to construct validity. We used code coverage metrics and redundancies in the test suite as our measure of high-quality. It is possible that other measures of test-suite quality could produce different results. However, we believe that our definition is valid and represents an important way to measure test suite quality. Another potential threat would be how the grades were assigned to programs. To reduce the chances of bias, the person who graded the assignments was not aware of the student's treatment group.

External validity: Because the student population in this academic program tend to have some professional programming experience, they may not be representative of other student populations. To increase external validity, we chose a series of assignments that would be common for a course of this level. Another potential threat is that the length of the semesters in which the studies were conducted was different due to a shortened summer term. To mitigate this threat, we kept the length of the time given for the assignments consistent between the two studies. Furthermore, the populations in the two studies were similar in terms of prior knowledge as validated by the pretests.

7 CONCLUSION AND FUTURE WORK

These outcomes of this study can be explained by the power of IBL-based *conceptual* feedback. IBL-based *conceptual* feedback informs the student about the underlying fundamental testing concepts that she or he did not test in her or his test suite, rather than providing the student with code coverage analytical feedback. This IBL type of feedback allows the student to determine on her or his own how to improve the test suite. The feedback also has the positive side-effects of helping the student gain knowledge, experience, and reinforcement of fundamental testing concepts, which. ultimately, will make the student better tester in the long term.

Testing Tutor's approach falls well within the purview of best pedagogy practices regarding providing students with the information to reach their learning objectives, rather than simply following the traditional right/wrong dichotomy of traditional testing coverage feedback. Testing Tutor trains students to think about testing in a specific and logical manner while still allowing them the opportunity to use their critical thinking skills to solve complex problems. From a pedagogical perspective, the results indicate that Testing Tutor can be used as an efficient modality to both analyze and reinforce testing concepts taught as part of the CS curriculum.

We plan additional development work for Testing Tutor including additional student and class analysis for the instructor, developing a plug-in that allows a student to submit their tests through an Integrated Development Environment (IDE), and additional user experience improvements. In addition, we also plan to perform additional empirical studies with the following objectives: 1) improve the feedback mechanisms 2) understand the effectiveness of Testing Tutor's feedback mechanisms at different levels of the curriculum 3) understand how Testing Tutor can be used as a tool for instructors to gauge learning and determine whether intervention is necessary to improve students' learning.

REFERENCES

[1] Sandra K. Abell. 2005. University Science Teachers as Researchers: Blurring the Scholarship Boundaries. *Research in Science Education* 35, 2-3 (2005), 281–298. https://doi.org/10.1007/s11165-004-5600-x

[2] Andrew Begel and Beth Simon. 2008. Struggles of new college graduates in their first software development job. *ACM SIGCSE Bulletin* 40, 1 (2008), 226–230. https://doi.org/10.1145/1352322.1352218

[3] Kevin Buffardi and Stephen H. Edwards. 2015. Reconsidering Automated Feedback: A Test-Driven Approach. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 416–420. https://doi.org/10.1145/2676723.2677313

[4] J. Collofello and K. Vehathiri. 2005. An Environment for Training Computer Science Students on Software Testing. *Proceedings Frontiers in Education 35th Annual Conference*. https://doi.org/10.1109/fie.2005.1611937

[5] D. M. de Souza, B. H. Oliveira, J. C. Maldonado, S. R. S. Souza, and E. F. Barbosa. 2014. Towards the use of an automatic assessment system in the teaching of software testing. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. 1–8.

[6] Stephen H. Edwards. 2003. Improving student performance by evaluating how well students test their own programs. *Journal on Educational Resources in Computing* 3, 3 (2003), 1–24. https://doi.org/10.1145/1029994.1029995

[7] Stephen H. Edwards. 2003. Teaching Software Testing: Automatic Grading Meets Test-First Coding. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '03)*. Association for Computing Machinery, New York, NY, USA, 318–319. https://doi.org/10.1145/949344.949431

[8] Stephen H. Edwards. 2004. Using software testing to move students from trial-and-error to reflection-in-action. *Proceedings of the 35th SIGCSE technical symposium on Computer science education - SIGCSE 04* (May 2004), 26. https://doi.org/10.1145/971300.971312

[9] Hisham Haddad. 2002. Post-graduate assessment of CS students: experience and position paper. *Journal of Computing Sciences in Colleges* 18, 2 (2002), 189–197.

[10] Robyn L. Houlden, Jamila B. Raja, Christine P. Collier, Albert F. Clark, and Jennifer M. Waugh. 2004. Medical students' perceptions of an undergraduate research elective. *Medical Teacher* 26, 7 (2004), 659–661. https://doi.org/10.1080/01421590400019542

[11] Helen H. Hu, Clifton Kussmaul, Brian Knaeble, Chris Mayfield, and Aman Yadav. 2016. Results from a Survey of Faculty Adoption of Process Oriented Guided Inquiry Learning (POGIL) in Computer Science. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. Association for Computing Machinery, New York, NY, USA, 186–191. https://doi.org/10.1145/2899415.2899471

[12] Christopher Justice, Wayne Warry, and Carl Cuneo. 2002. A grammar for inquiry: linking goals and methods in a collaboratively taught social sciences inquiry course. *Alan Blizzard Award* (2002), 15–27.

[13] Ayaan M. Kazerouni, Clifford A. Shaffer, Stephen H. Edwards, and Francisco Servant. 2019. Assessing Incremental Testing Practices and Their Impact on Project Outcomes. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 407–413. https://doi.org/10.1145/3287324.3287366

[14] Barbara Kitchenham, David Budgen, Pearl Brereton, and Philip Woodall. 2005. An investigation of software engineering curricula. *Journal of Systems and Software* 74, 3 (2005), 325–335.

[15] Timothy C. Lethbridge. 1998. A survey of the relevance of computer science and software engineering education. *Proceedings 11th Conference on Software Engineering Education*, 56–66. https://doi.org/10.1109/csee.1998.658300

[16] Timothy C. Lethbridge. 2000. Priorities for the education and training of software engineers. *Journal of Systems and Software* 53, 1 (2000), 53–71.

[17] Timothy C. Lethbridge. 2000. What knowledge is important to a software professional? *Computer (Long. Beach. Calif)* 33, 5 (2000), 44–50. https://doi.org/10.1109/2.841783

[18] Hanni Muukkonen, Minna Lakkala, and Kai Hakkarainen. 2005. Technology-Mediation and Tutoring: How Do They Shape Progressive Inquiry Discourse? *Journal of the Learning Sciences* 14, 4 (2005), 527–565. https://doi.org/10.1207/s15327809jls1404_3

[19] Raymond Pettit, John Homer, Roger Gee, Susan Mengel, and Adam Starbuck. 2015. An Empirical Study of Iterative Improvement in Programming Assignments. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 410–415. https://doi.org/10.1145/2676723.2677279

[20] Michael J. Prince and Richard M. Felder. 2006. Inductive Teaching and Learning Methods: Definitions, Comparisons, and Research Bases. *Journal of Engineering Education* 95, 2 (2006), 123–138. https://doi.org/10.1002/j.2168-9830.2006.tb00884.x

[21] James A. Shymansky, Larry V. Hedges, and George Woodworth. 1990. A reassessment of the effects of inquiry-based science curricula of the 60s on student performance. *Journal of Research in Science Teaching* 27, 2 (1990), 127–144. https://doi.org/10.1002/tea.3660270205

[22] Deborah A. Smith. 1996. *A Meta-Analysis of Student Outcomes Attributable to the Teaching of Science as Inquiry as Compared to Traditional Methodology*. Temple University.

[23] Jaime Spacco, David Hovemeyer, William Pugh, Fawzi Emad, Jeffrey K. Hollingsworth, and Nelson Padua-Perez. 2006. Experiences with marmoset. *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education - ITICSE 06* (2006). https://doi.org/10.1145/1140124.1140131

[24] Rachel Spronken-Smith, Tom Angelo, Helen Matthews, Billy O'Steen, and Jane Robertson. 2007. How Effective is Inquiry-Based Learning in Linking Teaching and Research? *An International Colloquium on International Policies and Practices for Academic Enquiry* 7, 4 (2007), 1–7.

[25] Lev S. Vygotskij and Michael Cole. 1981. *Mind in society: the development of higher psychological processes*. Cambridge: Harvard University Press.

[26] Taehyung (George) Wang, Diane Schwartz, and Robert Lingard. 2008. Assessing Student Learning in Software Engineering. *J. Comput. Sci. Coll.* 23, 6 (June 2008), 239–248. https://doi.org/10.5555/1352383.1352424

[27] Douglas N. Yarger, William A. Gallus, Michael Taber, J. Peter Boysen, and Paul Castleberry. 2000. A Forecasting Activity for a Large Introductory Meteorology Course. *Bulletin of the American Meteorological Society* 81, 1 (2000), 31–39. https://doi.org/10.1175/1520-0477(2000)081<0031:afafal>2.3.co;2

[28] R. Yin. 2006. Preparing Resource and Environmental Managers with International Understanding and Merits (PREMIUM): Introducing a research experience for undergraduates program. *Journal of Forestry* 104 (2006), 320–323.

[29] Uri Zoller. 1999. Scaling-up of higher-order cognitive skills-oriented college chemistry teaching: An action-oriented research. *Journal of Research in Science Teaching* 36, 5 (1999), 583–596. https://doi.org/10.1002/(SICI)1098-2736(199905)36:5<583::AID-TEA5>3.0.CO;2-M