DEVELOPMENT AND VALIDATION OF FEEDBACK-BASED TESTING TUTOR TOOL TO SUPPORT SOFTWARE TESTING PEDAGOGY

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Lucas Pascual Cordova

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Program:
Software Engineering

March 2020

Fargo, North Dakota

North Dakota State University
Graduate School

Title

DEVELOPMENT AND VALIDATION OF FEEDBACK-BASED TESTING TUTOR TOOL TO SUPPORT SOFTWARE TESTING PEDAGOGY

By

Lucas Pascual Cordova

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Dr. Gursimran Walia
Chair

Dr. Oksana Myronovych

Dr. Pratap Kotala

Dr. Limin Zhang

Approved:

3/12/2020
Date

Dr. Kendall Nygard
Department Chair

ABSTRACT

Current testing education tools provide coverage deficiency feedback that either mimics industry code coverage tools or enumerates through the associated instructor tests that were absent from the student's test suite. While useful, these types of feedback mechanisms are akin to revealing the solution and can inadvertently lead a student down a trial-and-error path, rather than using a systematic approach. In addition to an inferior learning experience, a student may become dependent on the presence of this feedback in the future. Considering these drawbacks, there exists an opportunity to develop and investigate alternative feedback mechanisms that promote positive reinforcement of testing concepts. We believe that using an inquiry-based learning approach is a better alternative (to simply providing the answers) where students can construct and reconstruct their knowledge through discovery and guided learning techniques. To facilitate this, we present Testing Tutor, a web-based assignment submission platform to support different levels of testing pedagogy via a customizable feedback engine. This dissertation is based on the experiences of using Testing Tutor at different levels of the curriculum. The results indicate that the groups using conceptual feedback produced higher-quality test suites (achieved higher average code coverage, fewer redundant tests, and higher rates of improvement) than the groups that received traditional code coverage feedback. Furthermore, students also produced higher quality test suites when the conceptual feedback was tailored to task-level for lower division student groups and self-regulating-level for upper division student groups. We plan to perform additional 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; and 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.

ACKNOWLEDGEMENTS

I would like to convey my deepest appreciation to my advisor, mentor, and hero, Dr. Gursimran Walia for his enthusiastic and encouraging support during the pursuit of my research. He consistently guided me when I needed assistance and provided inspiration and motivation that helped me complete this work. Dr. Walia taught me the overall process for performing quality research through continuous improvement. I was deeply inspired by the vision of Dr. Walia and Dr. Jeffrey Carver on this important body of discovery and research that led to my work.

I also wish to express my sincere regards to Dr. Oksana Myronovych and Dr. Pratap Kotala from the NDSU Computer Science Department and Dr. Limin Zhang from the NDSU Accounting and Information Systems Department for being on my supervisory committee and providing me quality feedback and guidance.

Finally, I would like to thank my friends and family, including my furry children, for providing support and comfort, and a special thanks to my husband, who supported me throughout the entire process by keeping me harmonious and helping me put pieces together. I will be forever grateful for your support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

TT..Testing Tutor.

LIST OF SYMBOLS

Δ.....Greek letter for Change

/ .....Per

1. INTRODUCTION

Software testing is a vital component in the development of high-quality software and can consume up to 50% of development effort [1]–[3]. In a study published by the National Institute of Standards and Technology (NIST), software engineers in the United States (U.S.) spend an average of 70-80% of their time testing and debugging, with the average bug taking 17.4 hours to fix, costing the U.S. economy over $50 billion annually [4]. Therefore, to be successful practitioners, students must master the ability to effectively test their software. Unfortunately, most students do not obtain sufficient testing skills while completing an undergraduate computer science (CS) degree [5], [6].

Consequently, CS students tend to take a trial-and-error approach to testing. An artificial harmony phenomenon is created upon a successful compilation or the comparison of the program's output to the instructor's solution. Software testing and the feedback mechanisms from existing state-of-the-art tools [7], [8] further encourage this behavior. The tools provide output-correctness information such as whether the instructor's tests have passed, diagnostic information illustrating the levels of code coverage achieved, and sometimes the exact portion of code containing the deficiencies.

It is believed that students are more successful at learning software testing principles using a reflection-in-action pedagogy [9]. A reflection-in-action pedagogy affects the process in which a student completes a task in the face of uncertainty. Rather than turning to trial-and-error, a student uses a reflective lens to examine the problem by recalling previously learned information, using new information, and reflecting on past experiences. Using this approach allows the student to generate new experiences while finding a viable solution when past experiences do not work in the new context without modification.

While many educators would agree that reflection-in-action is superior to trial-and-error, the typical software testing pedagogy artifacts (e.g. programming assignments), tools, and the feedback these tools produce are poor catalysts for promoting reflective behavior [9]. Upon submitting code to existing tools, students typically receive analytical feedback (e.g. raw code coverage), rather than conceptual feedback that would allow them to build reflective experiences. Using analytical feedback, a student is likely to correct the issue and move on without having built an experience from which to reflect on for future experiences [8]. Conversely, *conceptual feedback* informs the student which underlying fundamental testing concepts their test suites do not adequately cover and provides suggestions for the student to initiate their own learning process about those concepts.

Common testing mistakes by beginning programmers indicate that students lack an understanding of fundamental testing concepts [10]. Examples of these mistakes include: not testing a boundary condition, creating excess (redundant and unnecessary) test cases, missing a dimension in the data, not identifying risks, not being able to relate a test case to a risk, not considering side-effects, and poor ability to generalize results. To address these issues, we present Testing Tutor, a software testing education tool that can be integrated into any level of the CS curriculum. Testing Tutor's innovative contribution as a software testing education tool is the type of feedback it provides. Rather than providing the answer or specific issues, Testing Tutor provides conceptual feedback that reinforces the associated testing concepts to the student. This type of feedback helps a student build reflection-in-action experiences and in-turn results in a more concise and comprehensive test suite.

1.1. Research Goals

The goal of the Testing Tutor project is to help students learn testing concepts to enable them to become more effective software testers. Testing Tutor has been designed to automatically evaluate a student's test suite, provide feedback about its' completeness relative to the deficient fundamental testing concepts, and help students to improve their tests over time by allowing them to iteratively submit their test cases and receive feedback as well as access to learning materials to reinforce concepts.

The primary goal of this dissertation is defined as follows:

Validate the use of Testing Tutor to support software testing pedagogy across different levels of the curriculum.

1.2. Research Framework

To achieve the research goals listed in section 1.1, this dissertation followed a sequence of research and development activities described below:

I. Systematic Literature Review (SLR): The first step was to perform a SLR of software engineering and CS literature. This resulted in identification of the state-of-the-art software testing pedagogical tools and identifying their merits and shortcomings.

II. Development of Testing Tutor tool and feedback mechanisms: This step focused on developing the Testing Tutor tool using the Agile software development methodology. The initial version of Testing Tutor was configurable to three feedback treatments: 1) No feedback (primarily for pre/post test validation); 2) Detailed feedback (similar to raw/detailed code coverage from industry tools); and 3) Conceptual feedback.

III. Empirical validation and analysis of conceptual feedback: This step evaluated conceptual feedback versus raw/detailed feedback in Testing Tutor's *learning mode* at the junior level through studies over two quarter terms.

IV. Second iteration of development of Testing Tutor tool and improved feedback mechanisms: During this step, a new feedback framework was integrated into Testing Tutor to vary the levels of conceptual feedback. Enhancements and improvements were also implemented based on feedback from the end of study surveys from step III.

V. Empirical validation and analysis of new feedback framework: This step evaluated the feedback framework implemented in step IV at the CS2 level using Testing Tutor's *learning mode* and at the senior level in a project-based course using Testing Tutor's *development mode*.

2. SYSTEMATIC LITERATURE REVIEW

The literature review describes current testing educational approaches, testing educational tools, and support for pedagogical aspects to justify the need for Testing Tutor.

2.1. Testing Educational Approaches

This section describes three approaches educators have taken to address the shortcomings in testing education and how Testing Tutor differs from each.

2.1.1. Students Learn to Test by Submitting Test Cases along with Their Source Code

Bradshaw developed the Ante framework to automate the evaluation of student tests prior to submission of an assignment. The framework is based on Test-Driven-Development, a process in which a developer writes the test(s) prior to writing the code implementation. The instructor is required to develop tests, a valid implementation, and a bad implementation (for the purpose of validating good tests). In turn, a student writes tests and receives feedback whether the tests are correct. Upon having a set of valid tests, the student then develops an implementation and uses the system to see whether their implementation is correct.

The Ante framework resulted in higher-quality test cases, which resulted in higher-quality code [11]. A limitation was noted that the framework does add additional burden to the instructor in the form of grading test-cases [11]. Testing Tutor borrows the idea of encouraging students to write better test cases, but it does not add additional work for the instructors to manually grade the student test cases.

2.1.2. Students Learn to Test by Testing Code Written by Someone Else

The literature supports that students learn by testing code written by someone else [12]. These activities include 1) code reading to find a known number of faults [13]; 2) use of debugging tools [13]; 3) creating black box tests [12]; and 4) creating testing frameworks [12].

While these methods encourage students to test code more thoroughly, they do not provide feedback about why a test suite is incomplete. Testing Tutor borrows the idea of learning how to test by testing someone else's code. Unlike the aforementioned methods, Testing Tutor provides the student with conceptual feedback about why the test suite is incomplete. This helps the student learn fundamental testing concepts and how to systematically improve the test suite.

2.1.3. Students Learn to Test by Systematic Integration of Testing Across Multiple Courses

The Specification, Premeditation, Repeatability, Accountability, and Economy framework (SPRAE) developed by Jones [14] integrates testing across multiple courses as a means to unify a minimal set of test experiences, skills, and concepts. The framework initiates by providing students with the instructor's unit tests and then progressively enables the students to develop their own tests.

Testing Tutor is designed to be flexible enough for use across the curriculum. It allows a course instructor to tailor the type and amount of feedback depending upon what would be most helpful for that course. Testing Tutor also supports development of tests against a reference implementation (like early programming courses) or against the student's implementation (as the student starts developing and testing their own code).

2.2. Testing Education Tools

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

2.2.1. Collofello and Vehathiri

Collofello and Vehathiri developed a learning and training environment that enables students to develop knowledge and skills to perform requirement-based testing for beginning programming students (high-school and undergraduate). The environment consists of web-based

instructional materials and a simulator to test software programs. 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 the effectiveness of the test cases in finding faults [15]. The analytical data provided by the tool is like the data provided by industry code coverage tools (e.g. Code Cover, JaCoCo, coverage.py). At the conclusion of the testing exercise, the tool gives the students the 'answers' (e.g. the correct set of tests).

Conversely, Testing Tutor provides the students continuous feedback about which testing concepts were not fully exercised. It also provides the student access to learning materials to help them develop a better understanding of the concepts. The combination of feedback and resources helps the student improve their test suites.

2.2.2. Marmoset

Marmoset is an automated submission testing system designed to provide feedback in the form of incentivized access to the results of the instructor's tests on the students' code based on the assignment due date cycle [7]. The system has a staged test access and execution process, where it provides execution access to the public tests and the private tests after the public tests have all passed. Marmoset provides the instructor with feedback on the number of instructor's tests passed. The tool also captures code snapshots from the student to study the development process. At the end of the testing exercise, Marmoset also gives students the 'answers', similar to the system developed by Collofello and Vehathiri.

In contrast, Testing Tutor will facilitate learning by providing feedback on the student's own test suite based on a comparison with a reference test suite. Rather than providing feedback on which tests failed or which parts of the code are incorrect, Testing Tutor informs the student

which testing concepts were not fully exercised. This feedback mechanism focuses testing concepts and resources rather than the exact line or method containing the deficiency, which places the concepts and resources at the forefront of the feedback.

2.2.3. Web-CAT

The Web-based Center for Automated Testing tool (Web-CAT) [1, 7, 8] is a system that uses test coverage as a means of providing students with automated feedback on their code. Like Marmoset, Web-CAT uses the concept of staged test access and execution. The system provides detailed feedback on failed tests and provides improvement-focused code annotations. Web-CAT assesses the validity of the submitted code by comparing it against a reference solution, measuring the code coverage, and then running static analysis. Web-CAT integrates multiple tools and provides a raw dump of the data back to the student.

Web-CAT is similar to Marmoset as it provides feedback on failed tests, however, it provides the feedback in raw detailed dump form that has not been integrated with learning concepts. Consequently, Web-CAT's feedback is not conducive to learning. Raw dumps of information is difficult to interpret. The information must be correlated so that it can be deciphered. Testing Tutor takes a different approach. While Testing Tutor uses multiple components and tools to process the testing workflow, it provides synthesized feedback that is integrated with information and resources.

3. TESTING TUTOR

This section introduces Testing Tutor, its' main features, and ways it can be used to support testing pedagogy in programming.

3.1. Testing Tutor Introduction

Testing Tutor is a web-based software engineering education testing tool that is designed to help students learn to become more effective testers (achieving greater code coverage and fewer test redundancies). The tool identifies the testing concepts that are attributable to missing test cases by comparing them against either the reference implementation or the student's implementation. 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*, it helps a student test newly written code. Testing Tutor can be accessed at https://testingtutor.org. A demo account has been set up with the login: demo@testingtutor.org and password: RW9rT#U. Samples of feedback can be viewed by clicking "My Feedback". Sample feedback is also provided later in this chapter.

Testing Tutor is different from existing software testing education tools because of its' customizable feedback mechanism. The system is designed so that an instructor can tailor the level and type of feedback. This creates an opportunity for instructors and researchers to investigate feedback mechanisms that promote learning and improvement in software testing education. Testing Tutor can help students learn why their test suite is inadequate through test coverage data, while reinforcing key testing concepts.

3.2. Testing Tutor Features

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 – allows multiple institutions to set up and configure Testing Tutor for their respective institution. Segregates users, assignments, and reports per institution.
- Assignment repository – Assignments can be made private (to the instructor), shared with the institution, or publicly with other institutions.
- Tailored feedback – Instructor can select the following types of feedback: no feedback, detailed feedback, or conceptual feedback (including three types of conceptual feedback – task-level, process-level, and self-regulating-level).
- Course management – generate reports for courses, instructors, and individual students, and analysis for assignments or groups of assignments.
- Plug-in platform architecture – utilizes plug-in architecture to facilitate adding additional programming languages and is built to scale.

3.3. Usability of the System

The Assignment Portal allows instructors to search the repository for publicly shared assignments, assignments owned by their institution or private assignments based on criteria ranging from programming language, level-of-difficulty, and is searchable by tags as illustrated by Figure 1.

Figure 1: Assignment Portal

The assignment management page allows instructors to create, edit, and modify assignments. The instructor has the option of doing a 'preflight' on an assignment with a sample solution to preview the feedback. The instructor also has the option to view coverage and test redundancy analysis for an assignment or group of assignments as illustrated in Figure 2.

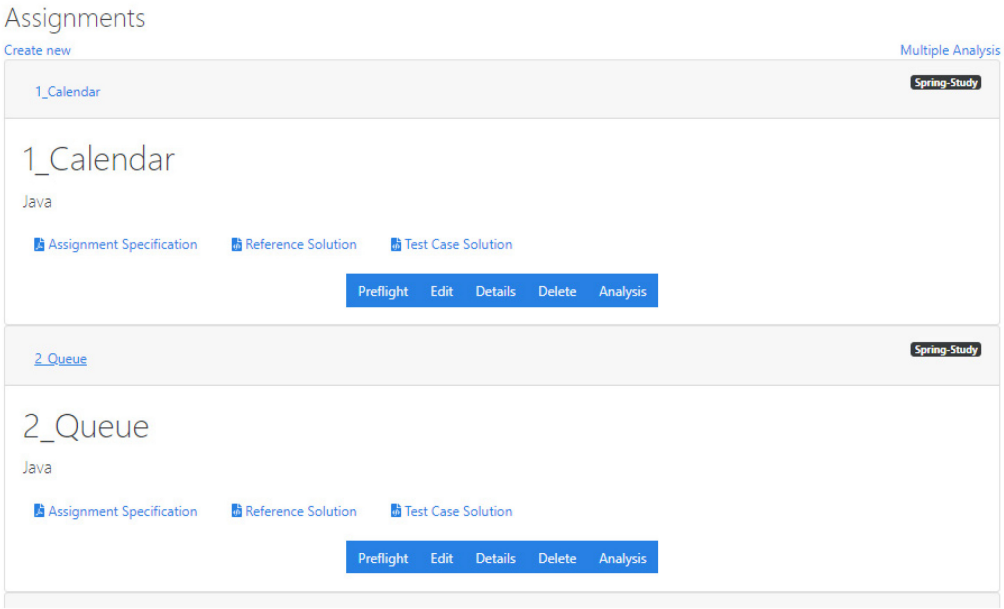

Figure 2: Assignment management (sample)

Analysis per assignment details all students' submissions over time which allows an instructor to view students' progress. The instructor can also download the data in comma-separated-value (CSV) format. An example of graphical analysis is shown in Figure 3.

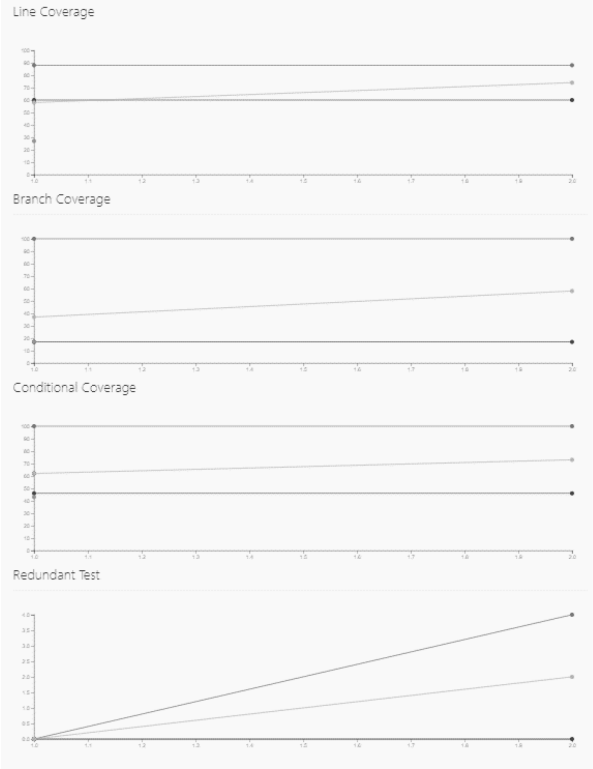

Figure 3: Analysis per assignment (sample)

Instructors can also group any number of assignments and view the average code coverage and redundant tests. This is useful for observing the overall trend for a group of assignments. The instructor also has the option of downloading the raw data in CSV format as depicted in Figure 4.

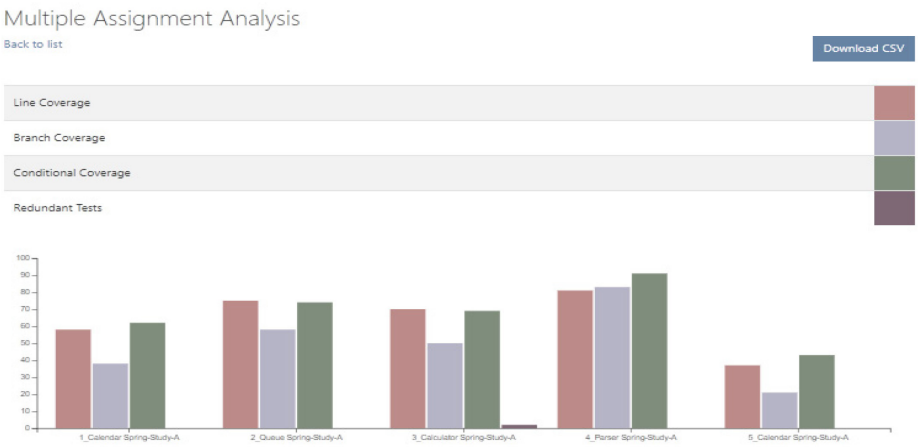

Figure 4: Assignment group analysis (sample)

The student submits their assignment and/or test code via a simple interface which allows them to select the application mode configured by the instructor. The student creates a ZIP archive of their test package folder containing their tests and submits it via the submission interface illustrated in Figure 5. If the instructor has enabled multiple application modes, the student is able to select which mode they would like Testing Tutor to use.

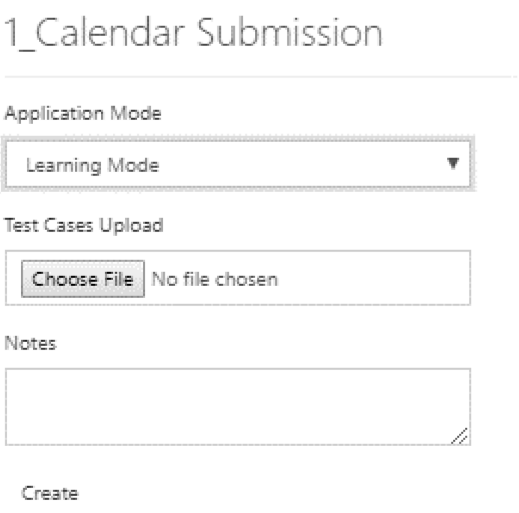

Figure 5: Student assignment submission for learning mode (sample)

Students with detailed feedback mode encounter a feedback report as seen in Figure 6. The feedback is similar to that received from coverage tools such as JaCoCo and CodeCover. In

addition, the student also receives information regarding the number of redundant tests. The student is able to view the data for any of their submissions.

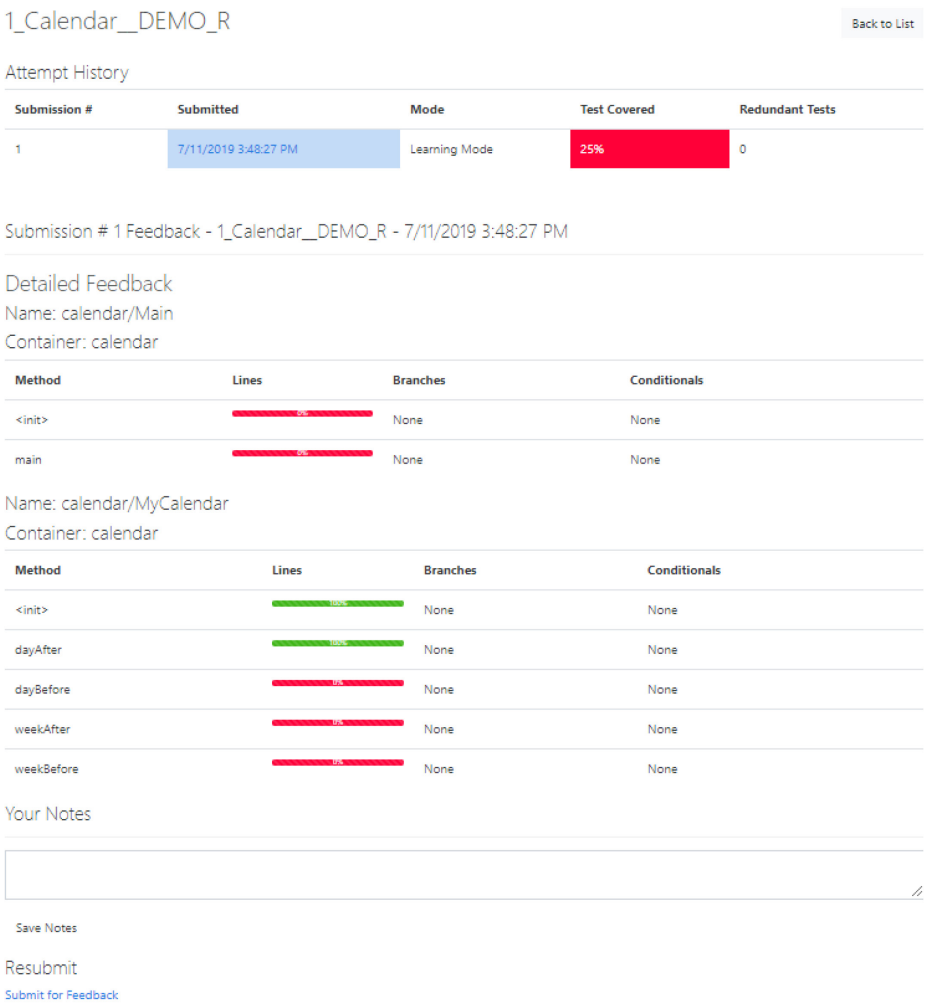

Figure 6: Detailed raw coverage and redundant tests feedback data (sample)

Students with conceptual feedback mode encounter feedback such as in Figure 7. The learning concepts missed are listed and highlighted along with the equivalence classes and additional resources for review.

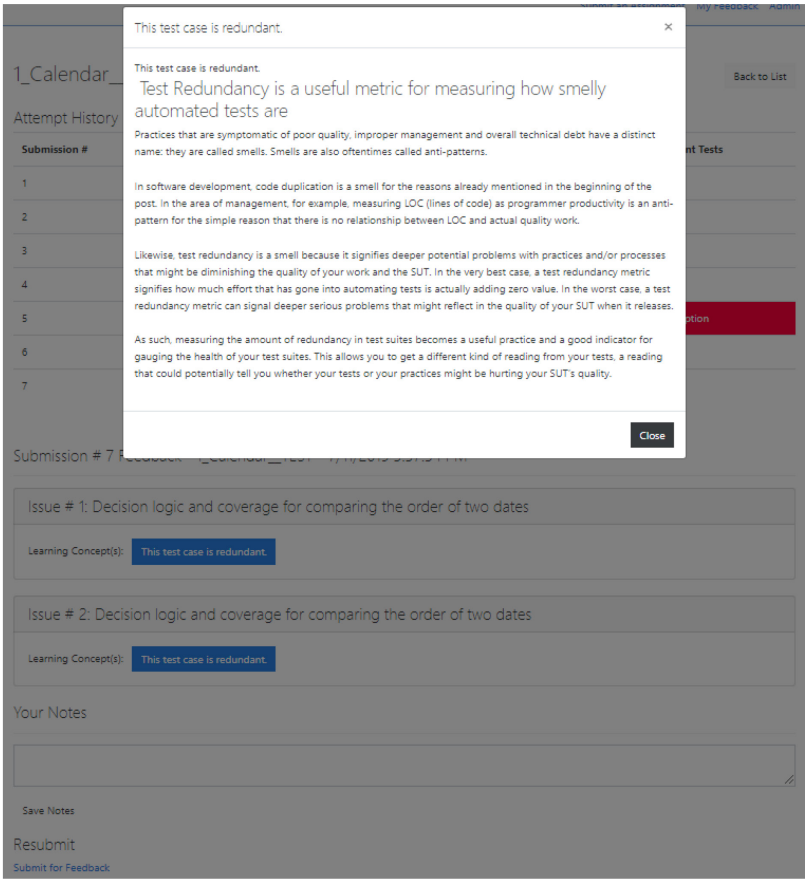

Figure 7: Student conceptual feedback with concept review for redundant tests (sample)

4. RESEARCH WORK

This chapter describes the research work completed. Sections 4.1 and 4.2 detail the parameters and results for two control group studies (Study 1 and Study 2) performed in CS3 using Testing Tutor in Learning Mode, where raw/detailed feedback was compared to conceptual feedback. Section 4.3 provides a discussion around the results of these preliminary studies. Section 4.4 discusses a feedback framework from the literature that was adapted to differentiate levels of conceptual feedback for the next studies. Section 4.5 details the parameters and results for a control group study in CS2 using Testing Tutor in Learning Mode, where two different levels of conceptual feedback treatments were compared. Section 4.6 details the parameters and results for a control group study in a software engineering senior project-based course using Testing Tutor in Development Mode, also comparing two different levels of conceptual feedback treatments. Table 1 summarizes the student-levels involved in each study, treatments, and Testing Tutor mode.

Table 1: Study summary

Study	Student-level	Treatments examined	TT Mode
1	CS3	Traditional raw/detailed vs. conceptual feedback	Learning Mode
2	CS3	Traditional raw/detailed vs. conceptual feedback	Learning Mode
3	CS2	Process-level vs. Self-regulating-level feedback	Learning Mode
4	Seniors project-based	Process-level vs. Self-regulating-level feedback	Development Mode

4.1. Study 1: A Control Group Study

Study 1 was designed to evaluate the impact that conceptual feedback has compared to raw/detailed code coverage feedback with respect to code coverage and the number of test redundancies. The study was conducted as a control group study in two sophomore-level software engineering classes taught by the same instructor. Each group used Testing Tutor for a pre-test, three programming assignments, and a post-test. Testing Tutor was configured specifically for each group for the type of feedback each group would receive. Group A was provided *traditional* raw/detailed code coverage feedback while Group B was provided *conceptual* feedback.

4.1.1. Study 1: Study Goal

The main goal of the study was to investigate and evaluate the impact that conceptual feedback has compared to traditional raw/detailed code coverage.

4.1.2. Study 1: Research Questions and Variables

This section describes the research question (RQ) and the variables used in the study. Table 2 lists the independent and dependent variables along with a description. The research question follows.

RQ: How do different types of feedback affect test code coverage, test redundancies, instructor's grade, and subjects' overall perception?

H*ypothesis*: Students will be able to obtain greater code coverage, reduce test redundancies, improve instructor's grade, and have a higher overall perception with conceptual feedback as compared to the detailed feedback.

Table 2: Study 1 - Variables

Independent Variables	***Description***
Treatment A: Traditional raw/detailed feedback	Traditional raw/detailed analytical code coverage feedback that is similar to the feedback provided by tools such as JaCoCo and CodeCover.
Treatment B: Conceptual feedback	Conceptual feedback which provides the student with the testing concepts that are not adequately tested in their tests, which includes resources to review (textual and video).
Treatment C: No feedback	No feedback provided.
Dependent Variables	***Description***
Code coverage percentage	The percentage of statement, branch, and conditional code coverage.
Number of redundant tests	The number of tests in the test suite that are considered redundant (cover code already tested by other tests).
Instructor's grade	The instructor's grade per assignment related to the quality of the test suite based on the rubric in Appendix B.
Number of submissions	The number of times each student submitted their tests cases to Testing Tutor.
Time between submissions	The amount of time that elapsed between each student's submissions to Testing Tutor.
Perception of student understanding of the feedback	The perception or rating of the student's understanding of the feedback provided.

4.1.3. Study 1: Artifacts

The following artifacts (programming assignments) were used in the study (Table 3).

Table 3: Study 1 - Artifacts

Artifact	*Description*
A	An I/O program written in Java 1.11 (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.
B	A state-based data structure abstract data type written in Java 1.11 (a queue).
C	An object-oriented calculator containing interfaces and inheritance written in Java 1.11.
D	A comma-separated-value (CSV) parser written in Java 1.11.

4.1.4. Study 1: Participating Subjects

The study was conducted in two sophomore software engineering courses at Oregon Institute of Technology which each had one section. One course had 25 students and the other course had 23 students (48 students in total). 28 out of 48 students participated in the study. Specifically, Group A contained 13 students and Group B contained 15 students.

4.1.5. Study 1: Study Procedure

The study procedure had seven sessions per group as detailed in Table 4. An initial session (Session 1) was spent training the students on Testing Tutor. A pre-test was given to both groups to gather a baseline on the students' ability to develop a comprehensive, yet small, test suite for Artifact A without any feedback (Treatment C) from Testing Tutor. The main part of the study involved the students developing a comprehensive, yet small, test suite for three programming assignments (Artifacts B, C, and D) with Group A receiving Treatment A feedback (detailed) and Group B receiving Treatment B feedback (conceptual). A post-test was then given to both groups to obtain a second baseline on the students' ability to develop a comprehensive, yet small, test suite for Artifact A without any feedback (Treatment C) from Testing Tutor. The

final part of the study included a survey that aimed to gather quantitative and qualitative feedback from the students' experience with the programming assignments, treatments, and usability of Testing Tutor.

Table 4: Study 1 - Procedure

Group A	*Group B*
Session 1 (30 min.) 26. Trained the students on Testing Tutor.	**Session 1 (30 min.)** 26. Trained the students on Testing Tutor.
Session 2 (75 min.) – Pre-Test 126. Assigned artifact A. 127. Set Testing Tutor to treatment C. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment C (no feedback)	**Session 2 (75 min.) – Pre-Test** 126. Assigned artifact A. 127. Set Testing Tutor to treatment C. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment C (no feedback).
Session 3 (75 min.) 126. Assigned artifact B. 127. Set Testing Tutor to treatment A. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment A feedback.	**Session 3 (75 min.)** 126. Assigned artifact B. 127. Set Testing Tutor to treatment B. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment B feedback.
Session 4 (75 min.) 126. Assigned artifact C. 127. Set Testing Tutor to treatment A. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment A feedback.	**Session 4 (75 min.)** 126. Assigned artifact C. 127. Set Testing Tutor to treatment B. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment B feedback.
Session 5 (75 min.) 126. Assigned artifact D. 127. Set Testing Tutor to treatment A. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment A feedback.	**Session 5 (75 min.)** 126. Assigned artifact D. 127. Set Testing Tutor to treatment B. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment B feedback.
Session 6 (75 min.) - Post-Test 126. Assigned artifact A. 127. Set Testing Tutor to treatment C. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment C (no feedback).	**Session 6 (75 min.) – Post-Test** 126. Assigned artifact A. 127. Set Testing Tutor to treatment C. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment C (no feedback).
Session 7 (20 min.) – Survey 26. Students were asked to complete a post-study survey.	**Session 7 (20 min.) – Survey** 26. Students were asked to complete a post-study survey.

4.1.6. Study 1: Data Collected

For each assignment submission, Testing Tutor collected the following data (dependent variables).

- *Code coverage* – The percentage of statement, branch, and conditional code coverage obtained.
- *Redundant tests* – The number of tests in the test suite that were considered redundant (code already tested by other tests).

In addition to the coverage metrics, the following additional data points were also later collected.

- *Instructor's grade* – The instructor's grade per assignment related to the quality of the test suite based on a rubric.
- *Number of submissions* – The number of times each student submitted their test cases to Testing Tutor.
- *Time between submissions* – The amount of time that elapsed between each student's submissions to Testing Tutor.
- *Perception of student understanding of the feedback* – An end-of-study optional and anonymous survey was conducted to gather the students' perception or rating of their understanding of the feedback provided as well as information regarding the usability of Testing Tutor.

4.1.7. Study 1: Summary of Results

This section describes the results that have been found in this study.

4.1.7.1. Pre-Test vs. Post-Test Results

To analyze the impact that the independent variables had on student learning, the dependent variables were collected for the pre-test and post-test. For the pre-test, it was hypothesized that there would not be a significant difference between groups with respect to the dependent variables suggesting that the groups were equally balanced in terms of their prior knowledge. For the post-test, it was hypothesized that there would be a significant difference between groups with respect to the dependent variables suggesting that the testing skills of Group B would improve using Treatment B.

To examine whether there was a statistical significance in the results, independent t-tests were conducted. Table 5 and Table 6 display the averages for each dependent variable along with the associated p-values. The results indicate relative average performance between groups for the pre-test, signifying that the groups were equally balanced in terms of their prior knowledge. This is further validated by the p-value for each dependent variable ($p > .05$). For the post-test, the results indicate a statistically significant difference ($p < .05$) between groups for

each dependent variable which validates that the testing skills of Group B improved using Treatment B.

Table 5: Dependent variable averages and p-values for Study 1 pre-test and post-test

Assignment	Dependent Variable	Group A Mean	Group B Mean	*p-value*
1 Pre-Test	Line Coverage	38%	33%	.93
	Branch Coverage	38%	37%	.93
	Conditional Coverage	38%	37%	.83
	# Redundant Tests	5.31	4.00	.20
	Instructor's Grade	57%	60%	.15
5 Post-Test	Line Coverage	36%	68%	.01
	Branch Coverage	43%	72%	.01
	Conditional Coverage	41%	73%	.01
	# Redundant Tests	3.77	2.20	.04
	Instructor's Grade	60%	80%	.01

Table 6: Pre-test vs. post-test statistical averages for Study 1

	Statistical Averages	Group A	Group B	*p-value*
Assignment 1 (Pre-Test)	Line Coverage Δ / Submission	1%	7%	.12
	Branch Coverage Δ / Submission	1%	7%	.12
	Conditional Coverage Δ / Submission	1%	6%	.20
	Redundant Tests Δ / Submission	4%	10%	.14
	Number of Submissions	4.23	2.33	.03
	Time between Submissions (min)	5.83	3.24	.04
Assignment 5 (Post-Test)	Line Coverage Δ / Submission	1%	8%	.06
	Branch Coverage Δ / Submission	2%	12%	.01
	Conditional Coverage Δ / Submission	1%	11%	.01
	Redundant Tests Δ / Submission	6%	9%	.71
	Number of Submissions	5.54	3.27	.01
	Time between Submissions (min)	6.86	4.73	.02

4.1.7.2. Study 1: Results from Study

Subjects in both groups completed Assignments 2, 3 and 4. Group A was assigned Treatment A (raw/detailed feedback). Group B was assigned Treatment B (conceptual feedback). It was hypothesized that there would be a significant difference in the average of the dependent variables between groups, suggesting the superiority of a treatment. To examine whether there was a statistical significance in the results, independent t-tests were conducted. Table 7 and Table 8 display the averages for each dependent variable per group and the corresponding p-values.

Table 7: Dependent variable averages and p-values for the main study

Assignment	**Dependent Variable**	**Group A Mean**	**Group B Mean**	***p-value***
2	Line Coverage	38%	55%	.03
	Branch Coverage	31%	49%	.01
	Conditional Coverage	39%	54%	.04
	# Redundant Tests	4.69	3.47	.15
	Instructor's Grade	58%	68%	.01
3	Line Coverage	43%	56%	.09
	Branch Coverage	39%	57%	.02
	Conditional Coverage	36%	55%	.01
	# Redundant Tests	4.23	3.73	.55
	Instructor's Grade	59%	69%	.01
4	Line Coverage	47%	61%	.01
	Branch Coverage	52%	55%	.63
	Conditional Coverage	51%	55%	.39
	# Redundant Tests	4.85	3.40	.09
	Instructor's Grade	63%	68%	.01

Table 8: Main study test statistical averages for Study 1

	Statistical Averages	Group A	Group B	*p-value*
Assignment 2	Line Coverage Δ / Submission	2%	4%	.30
	Branch Coverage Δ / Submission	2%	10%	.02
	Conditional Coverage Δ / Submission	1%	7%	.05
	Redundant Tests Δ / Submission	4%	11%	.15
	Number of Submissions	5.46	3.13	.01
	Time between Submissions (min)	6.83	4.91	.05
Assignment 3	Line Coverage Δ / Submission	2%	8%	.02
	Branch Coverage Δ / Submission	1%	7%	.02
	Conditional Coverage Δ / Submission	2%	6%	.15
	Redundant Tests Δ / Submission	7%	8%	.79
	Number of Submissions	5.15	2.73	.01
	Time between Submissions (min)	6.24	3.08	.01
Assignment 4	Line Coverage Δ / Submission	1%	10%	.04
	Branch Coverage Δ / Submission	1%	7%	.02
	Conditional Coverage Δ / Submission	1%	3%	.34
	Redundant Tests Δ / Submission	-2%	9%	.04
	Number of Submissions	3.92	2.93	.29
	Time between Submissions (min)	5.41	3.65	.18

While the raw averages and the number of test redundancies were higher overall for Group / Treatment B, the results indicate statistical significance ($p < .05$) per dependent variable varied by assignment as illustrated in Table 7 and Table 8. Average redundant tests were statistically insignificant across all assignments, suggesting that the treatments had no effect.

The instructor's grades improved for each assignment and were statistically significant across all assignments.

4.1.7.3. End of Study Survey

An optional end-of-study survey was given to subjects. The purpose of the survey was to gauge the study subject's perceptiveness of Testing Tutor. The survey contained eight quantitative questions that used a seven-point Likert-scale [1 = Entirely disagree; 2 = Mostly disagree; 3 = Somewhat disagree; 4 = Neither agree or disagree; 5 = Somewhat agree; 6 = Mostly agree; 7 = Entirely agree] followed by three qualitative questions. The survey questions are listed in Appendix A. 12 subjects in Group A (raw/detailed feedback) and 14 in Group B (conceptual feedback) completed the survey. An independent t-test was conducted to analyze whether the treatment had any effect on the subjects' perception of Testing Tutor. The results indicate that Treatment B (conceptual feedback) had a significant effect on the subjects' perception of Testing Tutor ($p < 0.05$), except for questions four and eight. Table 9 depicts the means per group and the p-values for each individual question.

Table 9: Study 1 mean per group and p-value per survey question

Question	*Dependent Variables*	*Group A Mean*	*Group B Mean*	*p-value*
1	The information that Testing Tutor provided helped me discover deficiencies in code coverage.	3.58	5.50	.007
2	The information Testing Tutor provided helped me discover redundant tests.	4.25	5.07	.174
3	The information Testing Tutor provided regarding code coverage deficiencies made a lasting impression on how I approach software testing in the future.	3.67	5.43	.036
4	The information Testing Tutor provided regarding redundant tests made a lasting impression on how I approach software testing in the future.	3.17	4.71	.052
5	Testing Tutor helped me become more EFFECTIVE at testing (achieving higher code coverage and reducing redundant tests)	3.33	5.43	.008
6	Testing Tutor helped me become more PRODUCTIVE at testing (achieving higher code coverage and reducing redundant tests during the amount of time spent).	3.75	5.79	.003
7	Testing Tutor is easy to use.	3.92	5.50	.047
8	I learned to use Testing Tutor quickly.	5.08	6.07	.100
9	I would recommend Testing Tutor to someone learning software testing	3.70	5.89	.004

The qualitative portion of the survey contained positive, negative, and constructive feedback for all three treatments. A consistent theme emerged regarding the lack of feedback (per design) in the pre-test and post-test. Several students commented on their inability to determine how to improve their code coverage or ensure they were not introducing redundant tests.

A positive theme that emerged with Treatment A was the color-coding of the coverage progress bars as well as the coverage bars themselves. Another positive theme was the ability to enter notes per submission which allowed the subjects to record the areas of focus for that submission and the ability to review them. A few comments centered around the lack of ability to click on coverage data and have the page transition to the source code containing the deficiencies like the method that some enterprise integrated development environments (IDEs) do (i.e. Microsoft Visual Studio).

Treatment B received praise regarding the conceptual review material, especially content that contained tangible examples (e.g. videos, code samples, etc.). Subjects liked the equivalence class information which helped clue them in to the area of code containing the deficiency. One area of frustration was around the lack of quantitative data. The feedback indicated that having code coverage data in addition to the conceptual data would have been ideal.

4.2. Study 2

Study 2 was designed to replicate and validate the results from study 1. A new artifact was added (Artifact E) to provide the students with a different program to complete in the post-test.

4.2.1. Study 2: Study Goal

The main goal of the study was to investigate and evaluate the impact that conceptual feedback has compared to traditional raw/detailed code coverage.

4.2.2. Study 2: Research Questions and Variables

This section describes the research question (RQ) and the variables used in the study. The independent and dependent variables are depicted in Table 10. The research question follows.

RQ: How do different types of feedback affect test code coverage, test redundancies, instructor's grade, and subjects' overall perception?

Hypothesis: Students will be able to obtain greater code coverage, reduce test redundancies, improve instructor's grade, and have a higher overall perception with conceptual feedback as compared to the detailed feedback.

Table 10: Study 2 - Variables

Independent Variables	***Description***
Treatment A: Traditional raw/detailed feedback	Traditional raw/detailed analytical code coverage feedback that is similar to the feedback provided by tools such as JaCoCo and CodeCover.
Treatment B: Conceptual feedback	Conceptual feedback which provides the student with the testing concepts that are not adequately tested in their tests, which includes resources to review (textual and video).
Treatment C: No feedback	No feedback provided.
Dependent Variables	***Description***
Code coverage percentage	The percentage of statement, branch, and conditional code coverage.
Number of redundant tests	The number of tests in the test suite that are considered redundant (cover code already tested by other tests).
Instructor's grade	The instructor's grade per assignment related to the quality of the test suite based on the rubric in Appendix B.
Number of submissions	The number of times each student submitted their tests cases to Testing Tutor.
Time between submissions	The amount of time that elapsed between each student's submissions to Testing Tutor.
Perception of student understanding of the feedback	The perception or rating of the student's understanding of the feedback provided.

4.2.3. Study 2: Artifacts

The artifacts (programming assignments) that were used in the study are shown in Table 11.

Table 11: Study 2 - Artifacts

Artifact	*Description*
A	An I/O program written in Java 1.11 (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.
B	A state-based data structure abstract data type written in Java 1.11 (a queue).
C	An object-oriented calculator containing interfaces and inheritance written in Java 1.11.
D	A comma-separated-value (CSV) parser written in Java 1.11.
E	A banking application written in Java 1.11.

4.2.4. Study 2: Participating Subjects

The study was conducted in two sophomore software engineering courses at Oregon Institute of Technology which each had one section. One course had 20 students and the other course had 20 students (40 students in total). 28 out of 40 students participated in the study. Specifically, Group A contained 15 students and Group B contained 16 students.

4.2.5. Study 2: Study Procedure

The study procedure had seven sessions per group as detailed in Table 12. An initial session (Session 1) was spent training the students on Testing Tutor. A pre-test was given to both groups to gather a baseline on the students' ability to develop a comprehensive, yet small, test suite for Artifact A without any feedback (Treatment C) from Testing Tutor. The main part of the study involved the students developing a comprehensive, yet small, test suite for three programming assignments (Artifacts B, C, and D) with Group A receiving Treatment A feedback

(detailed) and Group B receiving Treatment B feedback (conceptual). A post-test was then given to both groups to obtain a second baseline on the students' ability to develop a comprehensive, yet small test suite for Artifact E without any feedback (Treatment C) from Testing Tutor. The final part of the study included a survey that aimed to gather quantitative and qualitative feedback from the subjects' experience with the programming assignments, treatments, and usability of Testing Tutor.

Table 12: Study 2 - Procedure

Group A	*Group B*
Session 1 (30 min.) 26. Trained the students on Testing Tutor.	**Session 1 (30 min.)** 26. Trained the students on Testing Tutor.
Session 2 (75 min.) – Pre-Test 101. Assigned artifact A. 102. Set Testing Tutor to treatment C. 103. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 104. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. Testing Tutor provided treatment C (no feedback)	**Session 2 (75 min.) – Pre-Test** 101. Assigned artifact A. 102. Set Testing Tutor to treatment C. 103. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 104. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. Testing Tutor provided treatment C (no feedback).
Session 3 (75 min.) 101. Assigned artifact B. 102. Set Testing Tutor to treatment A. 103. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 104. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. Testing Tutor provided treatment A feedback.	**Session 3 (75 min.)** 101. Assigned artifact B. 102. Set Testing Tutor to treatment B. 103. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 104. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. Testing Tutor provided treatment B feedback.
Session 4 (75 min.) 101. Assigned artifact C. 102. Set Testing Tutor to treatment A. 103. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 104. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. Testing Tutor provided treatment A feedback.	**Session 4 (75 min.)** 101. Assigned artifact C. 102. Set Testing Tutor to treatment B. 103. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 104. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. Testing Tutor provided treatment B feedback.
Session 5 (75 min.) 101. Assigned artifact D. 102. Set Testing Tutor to treatment A. 103. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 104. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. Testing Tutor provided treatment A feedback.	**Session 5 (75 min.)** 101. Assigned artifact D. 102. Set Testing Tutor to treatment B. 103. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 104. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. Testing Tutor provided treatment B feedback.
Session 6 (75 min.) - Post-Test 126. Assigned artifact D. 127. Set Testing Tutor to treatment C. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment C (no feedback).	**Session 6 (75 min.) – Post-Test** 126. Assigned artifact D. 127. Set Testing Tutor to treatment C. 128. Instructed students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 129. Testing Tutor collected code coverage and the number of test redundancies after each student's submission. 130. Testing Tutor provided treatment C (no feedback).
Session 7 (20 min.) – Survey 26. Students were asked to complete a post-study survey.	**Session 7 (20 min.) – Survey** 26. Students were asked to complete a post-study survey.

4.2.6. Study 2: Data Collected

For each assignment submission, Testing Tutor collected the following data (dependent variables).

- *Code coverage* – The percentage of statement, branch, and conditional code coverage obtained.
- *Redundant tests* – The number of tests in the test suite that were considered redundant (code already tested by other tests).

In addition to the coverage metrics, the following additional data points were also later collected.

- *Instructor's grade* – The instructor's grade per assignment related to the quality of the test suite based on a rubric.
- *Number of submissions* – The number of times each student submitted their test cases to Testing Tutor.
- *Time between submissions* – The amount of time that elapsed between each student's submissions to Testing Tutor.
- *Perception of student understanding of the feedback* – An end-of-study optional and anonymous survey was conducted to gather the students' perception or rating of their understanding of the feedback provided as well as information regarding the usability of Testing Tutor.

4.2.7. Study 2: Summary of Results

This section describes the results that have been found in this study.

4.2.7.1. Pre-Test vs. Post-Test Results

To analyze the impact that the independent variables had on student learning, the dependent variables were collected for the pre-test and post-test. For the pre-test, it was hypothesized that there would not be a significant difference between groups with respect to the dependent variables suggesting that the groups were equally balanced in terms of their prior knowledge. For the post-test, it was hypothesized that there would be a significant difference between groups with respect to the dependent variables suggesting that the testing skills of Group B would improve using Treatment B.

To examine whether there was a statistical significance in the results, independent t-tests were conducted. Table 13 and Table 14 display the averages for each dependent variable along with the associated p-values. The results indicate relative average performance between groups for the pre-test, signifying that the groups were equally balanced in terms of their prior knowledge. This is further validated by the p-value for each dependent variable ($p > .05$). For the post-test, the results indicate a statistically significant difference ($p < .05$) between groups for each dependent variable which validates that the testing skills of Group B improved using Treatment B.

Table 13: Dependent variable averages for Study 1 pre-test and post-test

Assignment	Dependent Variable	Group A Mean	Group B Mean	*p-value*
1 Pre-Test	Line Coverage	33%	38%	.39
	Branch Coverage	35%	33%	.71
	Conditional Coverage	33%	36%	.51
	# Redundant Tests	4.47	5.75	.18
	Instructor's Grade	58%	57%	.40
5 Post-Test	Line Coverage	39%	70%	.01
	Branch Coverage	35%	67%	.01
	Conditional Coverage	48%	72%	.01
	# Redundant Tests	4.73	2.38	.01
	Instructor's Grade	60%	78%	.01

Table 14: Pre-test vs. post-test statistical averages for Study 2

	Statistical Averages	Group A	Group B	*p-value*
Assignment 1 (Pre-Test)	Line Coverage Δ / Submission	2%	6%	.18
	Branch Coverage Δ / Submission	2%	6%	.18
	Conditional Coverage Δ / Submission	1%	6%	.20
	Redundant Tests Δ / Submission	-1%	3%	.31
	Number of Submissions	4.80	2.56	.01
	Time between Submissions (min)	6.22	3.78	.03
Assignment 5 (Post-Test)	Line Coverage Δ / Submission	1%	13%	.01
	Branch Coverage Δ / Submission	2%	13%	.01
	Conditional Coverage Δ / Submission	2%	10%	.02
	Redundant Tests Δ / Submission	5%	12%	.13
	Number of Submissions	5.87	3.56	.01
	Time between Submissions (min)	6.89	3.75	.01

4.2.7.2. Study 2: Results from Study

Subjects in both groups completed Assignments 2, 3 and 4. Group A was assigned Treatment A (raw/detailed feedback). Group B was assigned Treatment B (conceptual feedback). It was hypothesized that there would be a significant difference in the average of the dependent variables between groups, suggesting the superiority of a treatment. To examine whether there was a statistical significance in the results, independent t-tests were conducted. Table 15 and Table 16 display the averages for each dependent variable per group and the corresponding p-values.

Table 15: Dependent variable averages for the main study

Assignment	**Dependent Variable**	**Group A Mean**	**Group B Mean**	***p-value***
2	Line Coverage	39%	57%	.01
	Branch Coverage	36%	53%	.01
	Conditional Coverage	43%	61%	.01
	# Redundant Tests	4.6	3.75	.27
	Instructor's Grade	60%	69%	.01
3	Line Coverage	43%	54%	.11
	Branch Coverage	52%	54%	.78
	Conditional Coverage	54%	58%	.55
	# Redundant Tests	5.93	2.63	.01
	Instructor's Grade	61%	69%	.01
4	Line Coverage	50%	48%	.65
	Branch Coverage	48%	49%	.83
	Conditional Coverage	47%	61%	.02
	# Redundant Tests	4.73	3.06	.03
	Instructor's Grade	62%	67%	.01

Table 16: Main study test statistical averages for Study 2

	Statistical Averages	Group A	Group B	*p-value*
Assignment 2	Line Coverage Δ / Submission	2%	12%	.01
	Branch Coverage Δ / Submission	1%	12%	.01
	Conditional Coverage Δ / Submission	2%	6%	.11
	Redundant Tests Δ / Submission	5%	11%	.10
	Number of Submissions	5.47	2.94	.01
	Time between Submissions (min)	6.75	3.82	.01
Assignment 3	Line Coverage Δ / Submission	1%	7%	.04
	Branch Coverage Δ / Submission	1%	9%	.01
	Conditional Coverage Δ / Submission	1%	10%	.01
	Redundant Tests Δ / Submission	3%	10%	.02
	Number of Submissions	4.93	3.00	.01
	Time between Submissions (min)	6.07	3.98	.04
Assignment 4	Line Coverage Δ / Submission	1%	6%	.10
	Branch Coverage Δ / Submission	1%	5%	.06
	Conditional Coverage Δ / Submission	1%	5%	.05
	Redundant Tests Δ / Submission	0%	15%	.01
	Number of Submissions	5.07	2.63	.01
	Time between Submissions (min)	6.45	3.93	.02

While the raw averages and the number of test redundancies were higher overall for Group / Treatment B, the results indicate that statistical significance ($p < .05$) per dependent variable varied by assignment as illustrated in Table 15. Average redundant tests were statistically significant across assignments, except for Assignment 2, suggesting that the

treatments influenced reducing test redundancies. The instructor's grades improved for each assignment and were statistically significant across all assignments.

4.2.7.3. End of Study Survey

An optional end-of-study survey was given to subjects. The purpose of the survey was to gauge the subjects' perceptiveness of Testing Tutor. The survey contained eight quantitative questions that used a seven-point Likert-scale [1 = Entirely disagree; 2 = Mostly disagree; 3 = Somewhat disagree; 4 = Neither agree or disagree; 5 = Somewhat agree; 6 = Mostly agree; 7 = Entirely agree] followed by three qualitative questions. The survey questions are listed in Appendix A. 15 participants in Group A (raw/detailed feedback) and 13 in Group B (conceptual feedback) completed the survey. An independent t-test was conducted to analyze whether the treatment had any effect on the subjects' perception of Testing Tutor. The results indicate that Treatment B (conceptual feedback) had a significant effect on the subjects' perception of Testing Tutor ($p < 0.05$) except for questions two and eight. Table 17 lists the means per group and the p-values for each individual question.

Table 17: Study 2 mean per group and p-value per survey question

Question	***Dependent Variables***	***Group A Mean***	***Group B Mean***	***p-value***
1	The information that Testing Tutor provided helped me discover deficiencies in code coverage.	3.56	6.00	.01
2	The information Testing Tutor provided helped me discover redundant tests.	4.25	5.23	.08
3	The information Testing Tutor provided regarding code coverage deficiencies made a lasting impression on how I approach software testing in the future.	2.29	3.81	.02
4	The information Testing Tutor provided regarding redundant tests made a lasting impression on how I approach software testing in the future.	3.19	4.85	.02
5	Testing Tutor helped me become more EFFECTIVE at testing (achieving higher code coverage and reducing redundant tests)	3.50	5.92	.01
6	Testing Tutor helped me become more PRODUCTIVE at testing (achieving higher code coverage and reducing redundant tests during the amount of time spent).	3.88	6.31	.01
7	Testing Tutor is easy to use.	3.89	5.85	.01
8	I learned to use Testing Tutor quickly.	5.19	6.23	.05
9	I would recommend Testing Tutor to someone learning software testing	3.72	5.75	.01

Like Study 1, the qualitative portion of the survey for Study 2 contained positive, negative, and constructive feedback for all three treatments. Contrary to the first study, participants did not mention any feedback regarding the lack of feedback in the pre-test and post-test. Similar positive feedback was provided regarding the color-coding of the coverage data in Treatment A and the conceptual review content provided by Treatment B. One common theme

emerged regarding frustration on the submission process. Subjects found the submission process cumbersome when compared to other code coverage tools built into modern IDEs. A subject suggested that Testing Tutor's feedback system should be integrated into a NetBeans plug-in that provided instant feedback.

4.3. Study 1 and Study 2 Discussion

The objective of these studies was to determine the merit of inquiry-based conceptual feedback over traditional feedback mechanisms in software testing education. The data and analysis provide insights into the effectiveness of inquiry-based conceptual feedback and on the dependent variables that we looked at. We now discuss the limitations to this study, insights, and possible implications for software testing education.

Undoubtedly, these studies have limitations. Although students experiencing conceptual feedback achieved higher code coverage, fewer test redundancies, and higher average improvement margins, statistical significance in the average variations varied by assignment in the main part of the study. It is unknown whether the assignment domains contributed to this. In addition, the instructor's solution was also based on one instructor's test case solution for each assignment. The solution to the assignment may have been biased towards certain testing concepts or may have varied in rigor. In future studies, it is recommended to peer-review the test cases and materials to ensure there is consensus on the concepts being exercised and the solutions.

Perhaps of greatest practical significance is that the data and analysis indicate that students that were equally balanced in terms of their prior knowledge (as validated by the pre-tests), were able to achieve significantly different levels of code coverage and test redundancies based on the feedback treatment. On average, students that experienced conceptual feedback

achieved 20% higher code coverage and 50% less test redundancies than students that experienced detailed feedback. From a pedagogical perspective, this indicates that Testing Tutor can be used as an efficient modality to both analyze and reinforce testing concepts. Furthermore, the statistical average rates of improvement in code coverage were on average 50% higher for students that experienced conceptual feedback as seen in the post-test analysis. In industry, this type of improvement would be a success as programmers would be writing higher quality code and increasing their overall productivity.

Another significant point is that students that experienced conceptual feedback spent 50% less time per submission, resulting in 50% fewer submissions than students that experienced detailed feedback. Students using conceptual feedback were able to correct the deficiencies in their test code in half the time and half of the submissions than students that experienced detailed feedback. The results support the notion that pointing a student in the right direction through conceptual feedback allows the student to make substantial progress faster. This also allows for instructors to focus their efforts on specific testing issues that might otherwise not be discussed due to time and energy that is focused on general testing.

These outcomes are explained by the power of inquiry-based conceptual feedback. Inquiry-based conceptual feedback informs the student about the underlying fundamental testing concepts that were not adequately tested in their test suite, rather than code coverage analytical feedback. This type of feedback allows the student to determine on their own how to improve the test suite with the positive side-effects of gaining additional knowledge, experience, and reinforcing the fundamental testing concepts. Ultimately, making the student a long-term better tester.

As mentioned in the results sections, the end-of-survey data indicated students' preference for conceptual feedback. The students that experienced conceptual feedback felt that Testing Tutor helped them meet the objectives of the assignments in a more productive and effective way compared to students that experienced detailed feedback. From a qualitative perspective, a theme emerged surrounding the conceptual review material, especially content that contained tangible examples (e.g. videos, code samples, etc.). Students liked the equivalence class information which helped clue them in to the area of code containing the deficiency. This falls well within the purview of best practices of providing the 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.

4.4. Feedback Framework for Studies 3 and 4

Motivated by the results from Study 1 and Study 2, this section discusses improving the conceptual feedback mechanisms via a feedback framework. The feedback framework is then evaluated in Study 3 and Study 4 with the following objectives: 1) improve the feedback mechanisms; and 2) understand the effectiveness of Testing Tutor's feedback mechanisms at different levels of the curriculum.

4.4.1. Objective 1: Improve the Feedback Mechanisms

The previous studies have shown that conceptual feedback when compared to traditional detailed feedback provides higher student learning of testing concepts. This work focused on further improving the feedback mechanisms provided by Testing Tutor so that higher student learning gains may be achieved. To further differentiate levels, a feedback framework from the

literature [18], which has been validated in computer science education [19], was applied for use in Testing Tutor and subsequently validated.

4.4.2. Objective 2: Manipulating Testing Tutor Feedback for Different Student Groups

The following focus of research was on evaluating and adapting Testing Tutor (and its' feedback mechanisms) to help students enrolled in both lower and upper division programming courses. Adapting and tailoring Testing Tutor feedback for different student groups provided additional empirical evidence on Testing Tutor's effectiveness.

4.4.3. Refinements for Conceptual Feedback

Hattie and Timperley [18] identified four levels of feedback based on a review of meta-analyses from studies in the literature. These feedback levels fall into a spectrum: task, process, self-regulation, and self. The levels and formal descriptions are listed in (Table 18). The "self-level" is often in the form of praise and has traditionally been used to comfort students, however it rarely contains task or process related information that helps students improve their learning. In fact, praise may be counterproductive and have negative consequences on students' self-evaluation of their ability and might be biased [18]. On the other hand, the "self-regulation level" contains the potential to encourage self-efficacy. Therefore, feedback on the self-level was not considered for the studies.

Table 18: Levels of feedback

Level	***Description***
Task	How well tasks are understood/performed.
Process	The main process needed to understand/perform tasks.
Self-regulation	Self-monitoring, directing, and regulating of actions.
Self	Personal evaluations and affect (usually positive) about the learner.

In addition to the four levels of feedback, Hattie and Timperley [18] proposed three major questions that must be addressed at each level so that the learner can recognize the gap between actual and desired performance and be able to act upon it. In order to be effective, feedback should include clear and sufficiently challenging objectives (Where am I going?), metrics or information that shows the gap between current and desired performance (How am I doing?), and have a clear roadmap of the best next steps to take (Where to next?). The three major questions and their descriptions are listed in (Table 19).

Table 19: Major questions to address when giving feedback

Question	***Description***
Where am I going?	Learning intention, goals, success criteria — goals need to be specific rather than general and sufficiently challenging.
How am I doing?	Actual performance, understanding — feedback regarding expected standard or success criteria and not in comparison with other students' progress.
Where to next?	Progression and new goals — information that leads to greater learning possibilities, enhanced challenges, and the development of more self-regulated learning.

Applying the Hattie and Timperley feedback framework to the conceptual framework feedback helped differentiate the conceptual feedback levels, while ensuring that the major questions were answered. This combination provided scholarly insight through empirical experimentation about which level(s) were most or least effective. A three-level conceptual feedback framework was established to map the feedback levels to conceptual feedback levels and answer the three questions accordingly. An example of translating the feedback framework to Testing Tutor software testing feedback for a fictious banking application under test is depicted in (Table 20). It should be noted that task-level feedback maps closely with traditional raw/detailed feedback.

Table 20: Translating Hattie and Timperley's feedback model to Testing Tutor example

Feedback Level	**TT Conceptual Level**	**Where am I going?**	**How am I doing?**	**Where to next?**
Task	Low	*Develop the most comprehensive, yet smallest test suite possible.*	TT provides: Test- Level pass/fail for each test case Sample feedback: *Branch on Line 22 is not covered.*	TT provides: Textual and video resources covering the concepts.
Process	Medium	*Develop the most comprehensive, yet smallest test suite possible.* *Boundary value*	TT provides: Equivalence class information for each test case that pass/fail Sample feedback: *Withdrawal penalty if overdrawn.*	TT provides: Textual and video resources covering the concepts. Apply again and resubmit Read more resources
Self-regulating	High	*Develop the most comprehensive, yet smallest test suite possible.*	TT provides: Deficient concepts list. Sample feedback: *Not all branches covered.*	TT provides: Textual and video resources covering the concepts.

4.5. Study 3: Understanding the Effect of Different Conceptual Levels in CS2

Previous studies focused on evaluating traditional raw/detailed feedback versus conceptual feedback. The next focus was on understanding the effect of different conceptual levels.

4.5.1. Study 3 Goal

The goal of this study was to evaluate the impact that different levels of conceptual feedback (based on the feedback framework presented in Section 4.4) can have on students' understanding of testing knowledge at the CS2-level. Specifically, process-level feedback will be compared to self-regulation-level feedback.

4.5.2. Study 3 Research Questions and Variables

This section describes the research question (RQ) and the variables used in the study. Table 21 lists the independent and dependent variables along with a description. The research question and hypothesis follow.

RQ: How do different levels of conceptual feedback affect test code coverage, test redundancies, instructor's grade, and subject's overall perception?

Hypothesis: A conceptual feedback treatment at the process-level will result in higher code coverage, fewer redundant tests, and higher average improvements per submission over information at the self-regulation level.

Table 21: Study 3 - Variables

Independent Variables	***Description***
Treatment A: Conceptual feedback at the process level	Conceptual feedback treatment which provides the student with feedback at the process-level. This will be the treatment for Group A.
Treatment B: Conceptual feedback at the self-regulation level	Conceptual feedback treatment which provides the student with feedback at the self-regulation-level. This will be the treatment for Group B.
Dependent Variables	***Description***
Code coverage percentage	The percentage of statement, branch, and conditional code coverage.
Number of redundant tests	The number of tests in the test suite that are considered redundant (cover code already tested by other tests).
Instructor's grade	The instructor's grade per assignment related to the quality of the test suite based on the rubric in Appendix B.
Number of submissions	The number of times each student submitted their tests cases to Testing Tutor.
Time between submissions	The amount of time that elapsed between each student's submissions to Testing Tutor.
Perception of student understanding of the feedback	The perception or rating of the student's understanding of the feedback provided.

4.5.3. Study 3 Artifacts

The following artifacts (programming assignments) were used in the study (Table 22). Students developed unit tests based on the program specifications.

Table 22: Study 3 - Artifacts

Artifact	*Description*
A	Lab 02 (pre-test with no TT feedback): Testing of Name Surfer (ranks name popularity over 100 years)
B	Lab 03: Testing of a Twitter hash-tag sentiment analyzer classification system
C	Lab 04: Testing of a currency exchange application
D	Lab 05: Testing of a JSON parser / object serializer
E	Lab 06 (post-test with no TT feedback): Testing of a Lyrics Analyzer

4.5.4. Study 3 Participating Subjects

The study was conducted in two sections of a CS2 computer science course at Western Oregon University. One section had 20 students and the other section had 22 students (42 students in total). All students opted to participate in the study.

4.5.5. Study 3 Procedure

The study procedure contained six steps per group as detailed in Table 23. An initial session was spent training the students on Testing Tutor. For each subsequent step, students were asked to develop a comprehensive, yet small, test suite for their assigned artifact by writing unit tests and submitting them to Testing Tutor for feedback as many times as they wish. The final part of the study included a survey that aimed to gather quantitative and qualitative feedback

from the subjects' experience with the programming assignments, treatments, and usability of Testing Tutor.

Table 23: Study 3 - Procedure

Steps
Step 1 • Train the students on Testing Tutor.
Step 2 • Assign artifact A. • Instructor collects each student's test cases and manually grades them.
Step 3 • Assign artifact B. • Set Testing Tutor to treatment for the group. • Instruct students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor.
Step 4 • Assign artifact C. • Set Testing Tutor to treatment for the group. • Instruct students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor.
Step 5 • Assign artifact D. • Set Testing Tutor to treatment for the group. • Instruct students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor.
Step 6 • Assign artifact E • Instructor collects each student's test cases and manually grades them.
Step 7 • Students will be asked to complete a post-study survey.

4.5.6. Study 3: Data Collected

For each assignment submission, Testing Tutor collected the following data (dependent variables).

- *Code coverage* – The percentage of statement, branch, and conditional code coverage obtained.

- *Redundant tests* – The number of tests in the test suite that were considered redundant (code already tested by other tests).

In addition to the coverage metrics, the following additional data points were also later collected.

- *Instructor's grade* – The instructor's grade per assignment related to the quality of the test suite based on a rubric in Appendix B.
- *Number of submissions* – The number of times each student submitted their test cases to Testing Tutor.
- *Perception of student understanding of the feedback* – An end-of-study optional and anonymous survey was conducted to gather the students' perception or rating of their understanding of the feedback provided as well as information regarding the usability of Testing Tutor.

4.5.7. Study 3: Summary of Results

This section describes the results that have been found in this study.

4.5.7.1. Pre-Test vs. Post-Test Results

To analyze the impact that the independent variables had on student learning, the dependent variables were collected for the pre-test and post-test. For the pre-test, it was hypothesized that there would not be a significant difference between groups with respect to the dependent variables suggesting that the groups were equally balanced in terms of their prior knowledge. For the post-test, it was hypothesized that there would be a significant difference between groups with respect to the dependent variables suggesting that the testing skills of Group A would improve using Treatment A.

To examine whether there was a statistical significance in the average between two groups, independent samples t-tests were conducted. Table 24 displays the averages for each dependent variable along with the associated p-values. The results indicate relative average performance between groups for the pre-test, signifying that the groups were equally balanced in terms of their prior knowledge. This is further validated by the p-value for each dependent variable ($p > .05$). For the post-test, the results indicate a statistically significant difference ($p < .05$) between groups for each dependent variable which validates that the testing skills of Group A improved using Treatment A (process-level feedback).

Table 24: Dependent variable averages for Study 3 pre-test and post-test

Assignment	**Dependent Variable**	**Group A Mean**	**Group B Mean**	***p-value***
2 (Pre-Test)	Line Coverage	73%	74%	.37
	Branch Coverage	73%	74%	.53
	Conditional Coverage	73%	74%	.68
	# Redundant Tests	4.45	4.90	.63
	Instructor's Grade	84%	84%	.54
6 (Post-Test)	Line Coverage	94%	85%	.01
	Branch Coverage	94%	85%	.01
	Conditional Coverage	94%	84%	.01
	# Redundant Tests	2.5	3.5	.15
	Instructor's Grade	95%	93%	.01

4.5.7.2. Study 3: Results from Study

Subjects in both groups completed Assignments 3, 4, and 5. Group A was assigned Treatment A (process-level feedback). Group B was assigned Treatment B (self-regulating-level feedback). It was hypothesized that there would be a significant difference in the average of the dependent variables between groups, suggesting the superiority of a treatment. To examine whether there was a statistical significance in the results, independent t-tests were conducted. Table 25 displays the averages for each dependent variable per group and the corresponding p-

values. Assignment 3 resulted in on-par performance between process-level and self-regulating conceptual feedback. Assignments 4 and 5 resulted in statistically significant performance for process-level feedback. Table 26 displays the statistical averages and the corresponding p-values. The results indicated that coverage change per submission was statistically significantly higher for process-level feedback versus self-regulating feedback. These results suggest that CS2-level students are able to achieve higher-quality test suites with the method-level conceptual feedback that process-level feedback provides.

Table 25: Dependent variable averages for Study 3

Assignment	**Dependent Variable**	**Group A Mean**	**Group B Mean**	***p-value***
3	Line Coverage	72%	69%	.05
	Branch Coverage	71%	70%	.62
	Conditional Coverage	70%	70%	.59
	# Redundant Tests	4.36	5.09	.05
	Instructor's Grade	85.4%	85.0%	.53
4	Line Coverage	74%	71%	.01
	Branch Coverage	77%	70%	.01
	Conditional Coverage	76%	68%	.01
	# Redundant Tests	2.39	3.65	.01
	Instructor's Grade	93.0%	84.2%	.01
5	Line Coverage	84%	74%	.01
	Branch Coverage	81%	73%	.01
	Conditional Coverage	82%	72%	.01
	# Redundant Tests	1.87	2.53	.04
	Instructor's Grade	94.1%	82.1%	.01

Table 26: Study test statistical averages for Study 3

	Statistical Averages	Group A	Group B	*p-value*
Assignment 3	Line Coverage Δ / Submission	2%	1%	.01
	Branch Coverage Δ / Submission	2%	1%	.01
	Conditional Coverage Δ / Submission	2%	1%	.01
	Redundant Tests Δ / Submission	1%	1%	.58
	Average Number of Submissions	11.05	12.55	.36
Assignment 4	Line Coverage Δ / Submission	2%	1%	.01
	Branch Coverage Δ / Submission	2%	1%	.01
	Conditional Coverage Δ / Submission	2%	1%	.01
	Redundant Tests Δ / Submission	1%	1%	.59
	Average Number of Submissions	12.95	12.65	.99
Assignment 5	Line Coverage Δ / Submission	2%	1%	.01
	Branch Coverage Δ / Submission	2%	1%	.01
	Conditional Coverage Δ / Submission	2%	1%	.01
	Redundant Tests Δ / Submission	1%	1%	.56
	Average Number of Submissions	10.85	13.82	.08

4.5.7.3. End of Study Survey

An optional end-of-study survey was given to subjects. The purpose of the survey was to gauge the study subject's perceptiveness of Testing Tutor. The survey contained eight quantitative questions that used a seven-point Likert-scale [1 = Entirely disagree; 2 = Mostly disagree; 3 = Somewhat disagree; 4 = Neither agree or disagree; 5 = Somewhat agree; 6 = Mostly agree; 7 = Entirely agree] followed by three qualitative questions. The survey questions are listed in Appendix A. 20 subjects in Group A (process-level feedback) and 22 in Group B (self-regulating-level feedback) completed the survey. An independent t-test was conducted to analyze whether the treatment had any effect on the subjects' perception of Testing Tutor. The

results indicate that Treatment A (process-level feedback) had a significant effect on the subjects' perception of Testing Tutor (where $p < 0.05$). Table 27 depicts the means per group and the p-values for each individual question. The survey results indicate that students felt more positively towards the use of Testing Tutor as a pedagogical tool when process-level feedback was provided versus self-regulating-level feedback. These results suggest again that process-level feedback is more appropriate for CS2-level students.

Table 27: Study 3 mean per group and p-value per survey question

Question	***Dependent Variables***	***Group A Mean***	***Group B Mean***	***p-value***
1	The information that Testing Tutor provided helped me discover deficiencies in code coverage.	5.85	3.50	.01
2	The information Testing Tutor provided helped me discover redundant tests.	5.35	4.20	.02
3	The information Testing Tutor provided regarding code coverage deficiencies made a lasting impression on how I approach software testing in the future.	5.70	3.70	.01
4	The information Testing Tutor provided regarding redundant tests made a lasting impression on how I approach software testing in the future.	4.90	3.10	.01
5	Testing Tutor helped me become more EFFECTIVE at testing (achieving higher code coverage and reducing redundant tests)	6.10	3.40	.01
6	Testing Tutor helped me become more PRODUCTIVE at testing (achieving higher code coverage and reducing redundant tests during the amount of time spent).	6.55	3.80	.01
7	Testing Tutor is easy to use.	5.75	3.80	.01
8	I learned to use Testing Tutor quickly.	6.15	5.10	.02
9	I would recommend Testing Tutor to someone learning software testing	6.00	3.30	.01

4.6. Study 4

Studies 1 through 3 were focused on lower division programming courses, where Testing Tutor was validated in Learning Mode. This study evaluated Testing Tutor's *development mode* functionality in an upper division course where students worked on their term-long projects. In Development mode, a student submits their solution to the assignment and their test cases and

receives the same type of customizable conceptual feedback, except based on comparison of the student's solution and test cases against the instructor's test cases.

4.6.1. Study 4 Goal: Evaluating Development Mode of Testing Tutor

The goal of Study 4 was to evaluate Testing Tutor in *development mode* as a pedagogical tool in an upper-division project-based course.

4.6.2. Study 4 Research Questions and Variables

This section describes the research question (RQ) and the variables used in the study. Table 28 lists the independent and dependent variables along with a description. The research question and hypothesis follow.

RQ: How do different levels of conceptual feedback affect test code coverage, test redundancies, instructor's grade, and subjects' overall perception?

Hypothesis: Students will be able to obtain greater code coverage, reduce test redundancies, improve instructor's grade, and have a higher overall perception with self-regulated-level conceptual feedback as compared to process-level conceptual feedback.

Table 28: Study 4 - Variables

Independent Variables	***Description***
Treatment A: Conceptual feedback at the process level	Conceptual feedback treatment which provides the student with feedback at the process-level. This will be the treatment for Group A.
Treatment B: Conceptual feedback at the self-regulation level	Conceptual feedback treatment which provides the student with feedback at the self-regulation-level. This will be the treatment for Group B.
Dependent Variables	***Description***
Code coverage percentage	The percentage of statement, branch, and conditional code coverage.
Number of redundant tests	The number of tests in the test suite that are considered redundant (cover code already tested by other tests).
Instructor's grade	The instructor's grade per assignment related to the quality of the test suite based on the rubric in Appendix B.
Number of submissions	The number of times each student submitted their tests cases to Testing Tutor.
Time between submissions	The amount of time that elapsed between each student's submissions to Testing Tutor.
Perception of student understanding of the feedback	The perception or rating of the student's understanding of the feedback provided.

4.6.3. Study 4 Artifacts

The following artifacts (programming assignments) will be used in the study (Table 29).

Table 29: Study 4 - Artifacts

Artifact	***Description***
A	Project Assignment – an e-commerce web site with REST APIs.

4.6.4. Study 4 Participating Subjects

The study was conducted at in a senior-level software engineering capstone course at Oregon Institute of Technology which has two sections. One section has 13 students and the other section has 11 students (24 students in total).

4.6.5. Study 4 Procedure

The study procedure will have three sessions per group as detailed in Table 30. An initial session will be spent training the students on Testing Tutor. The main part of the study will involve the students developing a comprehensive, yet small, test suite for a programming assignment with Testing Tutor in Development Mode with Group A receiving Treatment A feedback (process-level) and Group B receiving Treatment B feedback (self-regulating-level). The final part of the study will include a survey that aims to gather quantitative and qualitative feedback from the students' experience with the programming assignment, treatments, and usability of Testing Tutor.

Table 30: Study 4 - Procedure

Group A	Group B
Session 1 (30 min.) 1. Train the students on Testing Tutor.	Session 1 (30 min.) 1. Train the students on Testing Tutor.
Session 2 (40 days) 1. Assign artifact A. 2. Set Testing Tutor to treatment A. 3. Instruct students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 4. Testing Tutor collect code coverage and the number of test redundancies after each student's submission.	Session 2 (40 days) 1. Assign artifact A. 2. Set Testing Tutor to treatment B. 3. Instruct students to create the most complete, yet smallest test suite possible with the aid of Testing Tutor. 4. Testing Tutor collect code coverage and the number of test redundancies after each student's submission.
Session 3 (20 min.) – Survey 1. Students were asked to complete a post-study survey.	Session 3 (20 min.) – Survey 1. Students were asked to complete a post-study survey.

4.6.6. Study 4: Data Collected

For each assignment submission, Testing Tutor collected the following data (dependent variables).

- *Code coverage* – The percentage of statement, branch, and conditional code coverage obtained.
- *Redundant tests* – The number of tests in the test suite that were considered redundant (code already tested by other tests).

In addition to the coverage metrics, the following additional data points were also later collected.

- *Instructor's grade* – The instructor's grade per assignment related to the quality of the test suite based on a rubric in Appendix B.
- *Number of submissions* – The number of times each student submitted their test cases to Testing Tutor.
- *Perception of student understanding of the feedback* – An end-of-study optional and anonymous survey was conducted to gather the students' perception or rating of their understanding of the feedback provided as well as information regarding the usability of Testing Tutor.

4.6.7. Study 4: Summary of Results

Subjects in both groups completed the term project assignment. Group A was assigned Treatment A (process-level feedback) and Group B was assigned Treatment B (self-regulating-level feedback). It was hypothesized that there would be a significant difference in the average of the dependent variables between groups suggesting superiority of a treatment. To examine whether there was a statistical significance in the results, independent t-tests were conducted.

Table 31 displays the averages for each dependent variable per group and the corresponding p-values. Table 32 displays the statistical averages and the corresponding p-values. All dependent variables underscored support for the hypothesis that self-regulating-level feedback would be more appropriate for senior-level students. While the number of redundant tests were lower for self-regulating-level feedback, the result was not statistically significant. It was also noted that while the average change per dependent variable was higher for self-regulating-level feedback, they were not statistically significant.

Table 31: Dependent variable averages for Study 4

Assignment	**Dependent Variable**	**Group A Mean**	**Group B Mean**	***p-value***
Project	Line Coverage	74%	83%	.01
	Branch Coverage	73%	82%	.01
	Conditional Coverage	77%	83%	.05
	# Redundant Tests	4.77	4.67	.10
	Instructor's Grade	83%	90%	.01

Table 32: Main study test statistical averages for Study 4.

	Statistical Averages	**Group A**	**Group B**	***p-value***
Project	Line Coverage Δ / Submission	12%	7%	.19
	Branch Coverage Δ / Submission	12%	11%	.81
	Conditional Coverage Δ / Submission	7%	11%	.04
	Redundant Tests Δ / Submission	1%	1%	.57
	Average Number of Submissions	31.68	28.75	.46

4.6.8. End of Study Survey

An optional end-of-study survey was given to subjects. The purpose of the survey was to gauge the study subject's perceptiveness of Testing Tutor. The survey contained eight quantitative questions that used a seven-point Likert-scale [1 = Entirely disagree; 2 = Mostly disagree; 3 = Somewhat disagree; 4 = Neither agree or disagree; 5 = Somewhat agree; 6 = Mostly agree; 7 = Entirely agree] followed by three qualitative questions. The survey questions are listed in Appendix A. 13 subjects in Group A (process-level feedback) and 11 in Group B (self-regulating feedback) completed the survey. An independent t-test was conducted to analyze whether the treatment had any effect on the subjects' perception of Testing Tutor. The results indicate that Treatment B (self-regulating-level feedback) had an overall higher effect on the subjects' perception of Testing Tutor, with supporting statistical significance ($p < 0.05$) except for questions two, three, four, and eight. Table 33 depicts the means per group and the p-values for each individual question. The survey results indicate that students felt more positively towards the use of Testing Tutor as a pedagogical tool when self-regulating-level feedback was provided versus process-level feedback. These results suggest again that self-regulating-level feedback is more appropriate for upper-division level students.

Table 33: Study 4 mean per group and p-value per survey

Question	*Dependent Variables*	*Group A Mean*	*Group B Mean*	*p-value*
1	The information that Testing Tutor provided helped me discover deficiencies in code coverage.	4.00	6.09	.002
2	The information Testing Tutor provided helped me discover redundant tests.	4.54	5.18	.313
3	The information Testing Tutor provided regarding code coverage deficiencies made a lasting impression on how I approach software testing in the future.	4.15	5.64	.056
4	The information Testing Tutor provided regarding redundant tests made a lasting impression on how I approach software testing in the future.	3.54	4.91	.067
5	Testing Tutor helped me become more EFFECTIVE at testing (achieving higher code coverage and reducing redundant tests)	4.01	5.81	.022
6	Testing Tutor helped me become more PRODUCTIVE at testing (achieving higher code coverage and reducing redundant tests during the amount of time spent).	4.38	6.18	.014
7	Testing Tutor is easy to use.	4.31	5.92	.034
8	I learned to use Testing Tutor quickly.	5.39	6.27	.094
9	I would recommend Testing Tutor to someone learning software testing	4.27	6.08	.015

4.6.9. Study 3 and Study 4 Discussion

The objective of these studies was to investigate inquiry-based conceptual feedback in software testing education at different conceptual feedback levels based on the framework discussed in section 4.4 and at different levels of the curriculum. The data and analysis provide insights into the effectiveness of these levels of inquiry-based conceptual feedback and on the

dependent variables that were examined. Insights, possible implications for software testing education, and limitations to these studies will now be discussed.

Perhaps of greatest practical significance for Study 3 is that the data and analysis indicate that students that were equally balanced in terms of their prior knowledge (as validated by the pre-test), were able to achieve different levels of code coverage and test redundancies based on the conceptual feedback level that they were presented. Students that received process-level feedback achieved higher overall coverage on all three assignments, with statistical significance ($p < .05$) on assignments 4 and 5. The complexity of the assignments increased from assignment 3 to assignment 5. Given that the results on assignment 3 were not statistically significant, it is suggested that there is a threshold on complexity where process-level feedback has a higher impact. From a pedagogical perspective, the results of Study 3 indicate that CS2-level students can achieve higher quality test suites when provided with feedback at the process-level. The end-of-study survey for Study 3 showed higher support for process-level feedback ($p < .05$). This supports the theme that process-level feedback is more appropriate for lower-division classes.

The Study 4 results indicate that senior-level students achieved higher quality test suites with self-regulating-level feedback. This suggests that Testing Tutor in Development Mode is a viable pedagogical tool for learning software testing in a project-based work setting. While the number of submissions to Testing Tutor and the time between submissions varied in Study 4, improvement was made incrementally using both conceptual feedback levels, with self-regulating-level feedback being more effective for the senior-level students. The end-of-study survey for Study 4 showed higher averages for self-regulating-level feedback, though not all statistically significant. Since Study 4 was focused on upper-division students, self-regulating-

level conceptual feedback may have been more appropriate for these students because they have had some classes and experience with software testing, therefore the higher-level feedback coupled with accessible resources paired well for these students.

Study 3 and Study 4 have shown that the optimal conceptual feedback level may vary on the experience of the student. Lower-division students may need the level of detail that comes from process-level feedback. Upper-division students benefit more from the higher-level information that comes from self-regulating-level feedback.

Studies 3 and 4 do have some limitations. Assignment 3 in Study 3 showed that the dependent variables were on-par for process-level and self-regulating-level feedback, while Assignments 4 and 5 supported process-level. Assignment 3 may be an anomaly based on multiple factors, including the domain and complexity of the assignment and scope. During future studies, it may be beneficial to conduct a pre-study run on the assignments using a small subset of students (assuming the subset is representative of the larger group) or alternatively, a review with more faculty in order to gauge the appropriateness of the assignments. Furthermore, due to the logistics of the course, there was not a pre/post test conducted for Study 4. It is unknown whether the groups were equally balanced in knowledge prior to the study and whether a group's abilities improved following the use of Testing Tutor. Future studies with project-based assignments should integrate a pre/post test to address these limitations.

5. CONCLUSION

This section discusses the major contributions of the work described in this dissertation to computer science and software engineering education research and practice. Future work is also discussed.

5.1. Contribution to Research and Practice

The main goal of this dissertation is to enrich computer science education by deepening the understanding of feedback mechanisms for software testing. The results from Study 1 and 2 provide insights into learning through the standard feedback mechanism (raw/detailed coverage) versus conceptual feedback. The results from Study 3 and 4 provide insights into applying different levels of conceptual feedback at different levels of the curriculum. Study 4 demonstrated that Testing Tutor (in Development Mode) can be used as a pedagogical tool to support project-based assignments. These insights support that Testing Tutor had a positive impact on student learning and could be recommended for further treatment study and for use by other institutions.

The results of the studies provide insight into different feedback mechanisms in software testing. This work also contributes to focusing research effort on the improvement of these feedback mechanisms. These insights promote the use of Testing Tutor as a software engineering software testing education tool and opens opportunities for further research.

5.2. Grants under Review

1. National Science Foundation Level-2 grant for collaboration between North Dakota State University, University of Alabama, and Western Oregon University.

5.3. Future Work

In addition to the improvements to Testing Tutor, we also plan to perform additional empirical studies to continue 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; and 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. Additional development work for Testing Tutor includes 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. Expansion of the repository of reference programming assignments is also planned.

REFERENCES

[1] E. L. Jones, "An experiential approach to incorporating software testing into the computer science curriculum," *31st Annu. Front. Educ. Conf. Impact Eng. Sci. Educ. Conf. Proc. (Cat. No.01CH37193)*, pp. F3D-7-F3D-11, 2001.

[2] L. Osterweil, "Strategic directions in software quality," *ACM Comput. Surv.*, vol. 28, no. 4, p. 750, 1996.

[3] T. Shepard, M. Lamb, and D. Kelly, "More testing should be taught," *Commun. ACM*, vol. 44, no. 6, pp. 103–108, 2001.

[4] Research Triangle Institute, "The Economic Impacts of Inadequate Infrastructure for Software Testing. Planning Report 02-3," 2002.

[5] K. Buffardi and S. H. Edwards, "Reconsidering Automated Feedback: A Test-Driven Approach," *Proc. 46th ACM Tech. Symp. Comput. Sci. Educ. - SIGCSE '15*, pp. 416–420, 2015.

[6] R. E. Noonan and R. H. Prosl, "Unit testing frameworks," *Proc. 33rd SIGCSE Tech. Symp. Comput. Sci. Educ. - SIGCSE '02*, p. 232, 2002.

[7] J. Spacco, D. Hovemeyer, and W. Pugh, "Experiences with marmoset: designing and using an advanced submission and testing system for programming courses," *ITiCSE*, pp. 13–17, 2006.

[8] S. H. Edwards, "Improving student performance by evaluating how well students test their own programs," *J. Educ. Resour. Comput.*, vol. 3, no. 3, pp. 1–24, 2003.

[9] S. H. Edwards, "Using software testing to move students from trial-and-error to reflection-in-action," *Proc. 35th SIGCSE Tech. Symp. Comput. Sci. Educ. - SIGCSE '04*, no. May, p. 26, 2004.

[10] T. Wang, D. Schwartz, and R. Lingard, “Assessing student learning in software engineering,” *J. Comput. Sci. Coll.*, vol. 23, no. 6, pp. 239–248, Jun. 2008.

[11] M. K. Bradshaw, “Ante Up: A Framework to Strengthen Student-Based Testing of Assignments,” *Proc. 46th ACM Tech. Symp. Comput. Sci. Educ. - SIGCSE ’15*, pp. 488–493, 2015.

[12] D. Carrington and S. Kim, “Session S1C TEACHING SOFTWARE DESIGN WITH OPEN SOURCE SOFTWARE Session S1C,” *Education*, pp. 9–14, 2003.

[13] R. Chmiel and M. C. Loui, “Debugging: From Novice to Expert,” *LabVIEW für Einsteiger*, vol. 143, no. 1, pp. 69–74, 2019.

[14] E. L. Jones, “Software testing in the computer science curriculum -- a holistic approach,” *Proc. Australas. Conf. Comput. Educ. - ACSE ’00*, pp. 153–157, 2000.

[15] J. Collofello and K. Vehathiri, “An Environment for Training Computer Science Students on Software Testing,” in *Proceedings of the 35th Annual Frontiers in Education Conference*, 2006, pp. T3E-6-T3E-10.

[16] S. H. Edwards, “Teaching software testing,” *Companion 18th Annu. ACM SIGPLAN Conf. Object-oriented Program. Syst. Lang. Appl. - OOPSLA ’03*, no. May, p. 318, 2003.

[17] A. A. Callender and M. A. McDaniel, “The limited benefits of rereading educational texts,” *Contemp. Educ. Psychol.*, vol. 34, no. 1, pp. 30–41, 2009.

[18] J. Hattie and H. Timperley, “The power of feedback,” *Rev. Educ. Res.*, vol. 77, no. 1, pp. 81–112, 2007.

[19] C. Ott, A. Robins, and K. Shephard, “Translating principles of effective feedback for students into the CS1 context,” *ACM Trans. Comput. Educ.*, vol. 16, no. 1, pp. 1–27, 2016.

APPENDIX A. SURVEY QUESTIONS FOR STUDY 1 AND STUDY 2

Testing Tutor Survey

Please answer the following questions.

[1 = Entirely disagree; 2 = Mostly disagree; 3 = Somewhat disagree; 4 = Neither agree nor disagree; 5 = Somewhat agree; 6 = Mostly agree; 7 = Entirely agree]

1. I was in Group:
    a. A
    b. B
2. The information Testing Tutor provided helped me discover deficiencies in code coverage.
3. The information Testing Tutor provided helped me discover redundant tests.
4. The information Testing Tutor provided regarding code coverage deficiencies made a lasting impression on how I approach software testing in the future.
5. The information Testing Tutor provided regarding redundant tests made a lasting impression on how I approach software testing in the future.
6. Testing Tutor helped me become more EFFECTIVE at testing (achieving higher code coverage and reducing redundant tests).
7. Testing Tutor helped me become more PRODUCTIVE at testing (achieving higher code coverage and reducing redundant tests during the amount of time spent).
8. Testing Tutor is easy to use.
9. I learned to use Testing Tutor quickly.
10. I would recommend Testing Tutor to someone learning software testing.

11. What were some of the most powerful learning moments in using Testing Tutor and what them so?
12. What were some of the most challenging moments and what made them so?
13. Is there any other feedback that you have?

APPENDIX B. ASSIGNMENT RUBRIC

Type / Coverage %	**0%-50%**	**51% - 60%**	**61% - 70%**	**71% - 80%**	**81% - 94%**	**> 95%**
Line Coverage points	0	70	75	85	95	100
Branch Coverage points	0	70	75	85	95	100
Conditional Coverage points	0	70	75	85	95	100

Measure / # Redundant Tests	**0**	**1.0 - 2.0**	**2.1 - 4.0**	**4.1 - 7.0**	**> 7.1**
# Redundant Tests points	100	95	85	75	0