# An Experience-Driven Pedagogy for the Instruction of Software Testing in Computer Science

Christopher Duane Brown
*Michigan Technological University*

Follow this and additional works at: https://digitalcommons.mtu.edu/etds
Part of the Computer Sciences Commons

# AN EXPERIENCE-DRIVEN PEDAGOGY FOR THE INSTRUCTION OF SOFTWARE TESTING IN COMPUTER SCIENCE

By

Christopher Duane Brown

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY

2013

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Computer Science.

Department of Computer Science

| | |
|---|---|
| Dissertation Advisor: | Dr. Robert Pastel |
| Committee Member: | Dr. Linda Ott |
| Committee Member: | Dr. Charles Wallace |
| Committee Member: | Dr. Marika Seigel |
| Department Chair: | Dr. Charles Wallace |

To my family, my teachers and my friends

who inspired me to learn, think and never stop striving for my potential. Without you, I could not be the person I am today.

# Contents

# List of Figures

## Acknowledgments

This research could not have been done without the many people who supported me:

- † My wife, Sara, for providing strength when I was weak and encouragement when I needed it most.
- † My parents for supporting my decision to strive for my academic potential.
- † My advisor, Dr. Robert Pastel, who gave me guidance to find my own path.
- † My Ph.D. committee and my master's committees, who challenged me to discover.
- † The Computer Science students at Michigan Technological University who participated in my research.
- † The SIGCSE community, who provided a forum for educational research in Computer Science.
- † The scientists and others working with us in the Citizen Science project.
- † Finally, I would like to thank all my friends for being the sounding boards of my thoughts and energy.

# Abstract

In the realm of computer programming, the experience of writing a program is used to reinforce concepts and evaluate ability. This research uses three case studies to evaluate the introduction of testing through Kolb's Experiential Learning Model (ELM). We then analyze the impact of those testing experiences to determine methods for improving future courses.

The first testing experience that students encounter are unit test reports in their early courses. This course demonstrates that automating and improving feedback can provide more ELM iterations. The JUnit Generation (JUG) tool also provided a positive experience for the instructor by reducing the overall workload.

Later, undergraduate and graduate students have the opportunity to work together in a multi-role Human-Computer Interaction (HCI) course. The interactions use usability analysis techniques with graduate students as usability experts and undergraduate students as design engineers. Students get experience testing the user experience of their product prototypes using methods varying from heuristic analysis to user testing. From this course, we learned the importance of the instructors role in the ELM.

As more roles were added to the HCI course, a desire arose to provide more complete, quality assured software. This inspired the addition of unit testing experiences to the course. However, we learned that significant preparations must be made to apply the ELM when students are resistant.

The research presented through these courses was driven by the recognition of a need for testing in a Computer Science curriculum. Our understanding of the ELM suggests the need for student experience when being introduced to testing concepts. We learned that experiential learning, when appropriately implemented, can provide benefits to the Computer Science classroom. When examined together, these course-based research projects provided insight into building strong testing practices into a curriculum.

# Chapter 1

# Experiential Learning

I hear and I forget, I see and I remember, I do and I understand.

-Confucius

Idioms like this reflect the perception of experience as a tool in the path to knowledge. Methods of experience, such as apprenticeships, have been used throughout history to train people for their roles in society. Aristotle called the knowledge gained through experience *techne*, or productive knowledge. It is one of the three parts of the Aristotlean epistemological taxonomy, with theoretical (*episteme*) knowledge and practical (*praxis*) knowledge. The word is also the root of *technology*, as Johnson explains while describing the technological necessity of productive knowledge [37].

In the realm of computer programming, students are being taught how to develop new technology through programming. The ultimate goal of an undergraduate curriculum for a computer programmer is the ability to create technology, as they will in the software industry. In industry, programmers must continually invent new technology; it is wasteful to reinvent what can be copied or purchased. However, in academia, students often repeat exercises so that they can be evaluated in the process of becoming productive programmers. Early courses for programmers use code labs, program assignments, and written pseudo-code on exams to provide practice and demonstration of their abilities. These techniques rely on the concept that, with direction and knowledge from the classroom, experience provides students the understanding that they need to develop as programmers.

In software development, testing is used to verify software quality. Students often use manual tests to ensure that their software works as intended. However, the professional environment has recently increased the use of automated testing tools dramatically [70].

The focus of this research is to establish the extent that the computer science classroom can benefit by including new testing experiences. We will also explore how adding testing practices can improve the classroom experiences which students currently encounter. In addition, we will explore some instructor techniques that could be used to enhance those experiences. Finally, we will discuss the difficulties that can arise and methods for mitigating those difficulties.

## Experiences as a Lecturer

My personal experiences as an instructor shaped the nature of the educational research done here. Described are the courses that I taught, and the main projects completed within those courses.

In my Master's thesis I explored cooperative learning in an introductory programming course [12, 46]. The course was a Visual Basic programming course in the Information, Science and Technology department at the University of Missouri-Rolla. The experiment was intended to determine if cooperative techniques applied to paired programming exercises. The two benchmarks were case studies on a mathematics course and an accounting course. The mathematics course showed significant benefit from cooperative techniques, while the accounting course did not [33, 73]. Like the accounting class, the results of our cooperative programming course showed no significant improvement for student learning. Based on the work of D.W. Johnson, et al. [36] we determined that, like an accounting course, students without experience in computer programming need to gain core ideas independently [12, 46].

The necessity of core ideas being gained independently parallels Robert Johnson's discussion of the uncertainty of productive knowledge. Johnson distinguishes the creative aspect of *techne* from Plato's description of certainty within theoretical *episteme* knowledge. The production of technology is uncertain, since its successfulness is determined by its use [37]. If the core ideas were certain, like *episteme* knowledge, they could be taught in the classroom and the labwork would simply be a demonstration. In this case, we would expect the cooperative work to follow the mathematics course. However, since the course paralleled the accounting class, programming falls into the uncertain *techne* knowledge.

Soon after beginning PhD work at Michigan Technological University, my interest and experience in Human Computer Interaction (HCI) inspired the development of a graduate level HCI course [13]. This course was combined with the existing undergraduate course [61]. The primary focus of the course was to provide the industrial experience of a usability expert for graduate students. This reflects the intention of students taking the course as part of a post-graduate degree emphasizing usability, either industrially or academically. To provide a concrete experience, the graduate students evaluate projects created in the undergraduate course. The undergraduate students gained experience designing usable interfaces. In addition, the course provided unfabricated examples of user interface design and demonstrated the relationship with developers.

In my second position as a lecturer, I continued to emphasize the need for increased experience. That course was a sophomore level data structures course, as described by Pastel [62]. This course included five complex programming assignments intended to demonstrate competence. Some students had difficulty grasping the concepts of the data structures and the sample application. To apply this knowledge to the data structures course, we split each assignment into two parts. The first assignment provided feedback and direction for the second, the third for the fourth, and so on. Due to the time constraints of providing feedback for more assignments, a series of automated grading techniques were developed, which ultimately led to the JUnit Generation (JUG) automated grading system.

After developing the JUG for the Data Structures course [15], I began to teach an accelerated introductory programming course. The course is designed for students who had done basic programming, but had no formal, college equivalent coursework. This was an accelerated course and students expected a higher pace; therefore I included 14 programming exercises through the semester. Using the JUG grading tools, feedback for these assignments was provided quickly. At my suggestion, a lab was added to increase the quantity of experiences in the course. To account for the change in structure, the original 14 homework assignments were split into 10 laboratory assignments and 11 homework assignments.

The model of my computer science pedagogy is shaped by the courses that I designed and taught. Each course provided me an experience teaching students and time to reflect on that experience. Then I could read academic literature and abstract those experiences into a pedagogical model. Based on that growing pedagogical model, I could develop methods for iteratively improving the courses. The natural cycle of experience, reflection, abstraction and improvement is the core of the Experiential Learning Model described by David Kolb.

## 1.1 The Experiential Learning Model

David Kolb combined the theories of of three major experience based approaches to education to create the Experiential Learning Model (ELM) [42]. In the ELM, he considers four mental activities taken by the learner, and discusses the cycle of learning that develops from those stages. Kolb identifies the four stages as:

1. Concrete Experience (CE)
2. Reflective Observation (RO)
3. Abstract Conceptualization (AC)
4. Active Experimentation (AE)

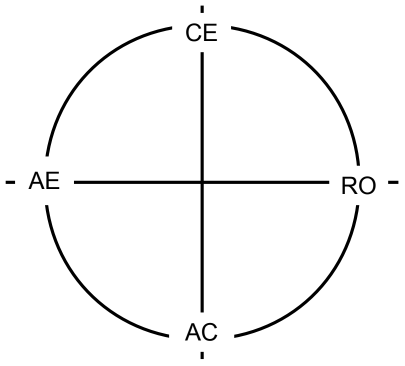


**Figure 1.1:** ELM with two axes.

Kolb also indicates that although the cycle naturally evolves from other work, there is an implied conflict of abilities. The learner must be able to both take action (AE) and pause to reflect (RO) on each experience. They must also be able to treat experiences as both concrete instances (CE) and theoretical generalizations (AC). To represent these conflicting abilities, he has two axes in the ELM diagram. See Figure 1.1. As a person moves through the process, they may be acting in one direction more than another, but will move about in a roughly clockwise direction.

In addition, Kolb emphasizes the level of feedback as an important component of the

ELM. Without appropriate feedback, there is an imbalance along the observation/action axis (AE—RO). This means that the learner is either focused on taking actions and does not (or cannot) observe feedback, or that they are focused on attempting to observe, that they refuse to take new actions.

One of Kolb's key points combines the Lewinian learning cycle with Dewey's idea of model development [42]. Each person has a model of an object or activity, which could be simplistic or incorrect. The ELM expects the cycle to begin with an experience that either supports, expands or replaces the learners current model. During the critical abstraction phase, a learner may be confronting an experience that strongly contradicts their current model. If this is the case, the learner may replace their current model with the new one, or the learner may regress [42]. This *model regression* can cause the learner to ignore the experience, or even simplify their model to one that they had rejected earlier. Although model regression could be beneficial in some situations, in an academic environment it could hinder learning.

Expanding on the idea of model regression, Kolb emphasizes the necessity of the learner mentally participating in the experience. Without that participation, a learner cannot update their mental model, and therefore, cannot learn. Kolb says this:

> "[Learners] must be able to involve themselves fully, openly, and without bias in new experiences (CE). They must be able to reflect on and observe

their experiences from many perspectives (RO). They must be able to create concepts that integrate their observations into logically sound theories (AC), and they must be able to use these theories to make decisions and solve problems (AE)." [42]

## 1.2 Research Questions

If we are to introduce testing experiences in the classroom for students, we must be able to relate the usefulness of those experiences. Unlike academic environments, where testing can be rare, industry requires testing as part of most quality assurance processes. Software engineers are expected to know how to verify their code and are expected to generate automated test code as part of their development process. Meanwhile, companies continue to balance quality assurance against test cost [39, 70]. This is not limited to profit-driven software development; open source projects such as Apache and Mozilla follow similar competitive programming practices, including large amounts of automated testing [57]. These engineering environments have one significant advantage over the academic environment: the developers have years of customer-driven experiences which guide their engineering decisions.

In order to provide academic experiences which provide similar benefits to the industrial experiences, we have chosen to use Kolb's Experiential Learning Model as the basis of comparison. The following research questions help us determine the capabilities and restrictions when attempting to replicate industrial experiences.

**How can the Experiential Learning Model (ELM) be used to incorporate testing experiences to computer programming courses?**

Computer programming requires a combination of abstract and concrete problem solving skills. As discussed earlier, these skills can be difficult to incorporate together [12, 46]. In addition, good testing practices can be difficult for students to grasp [69]. By adding testing experiences into courses at different levels, we can determine methods for applying the ELM to testing.

**When introducing testing experiences, what additional steps must be taken to ensure a positive learning experience?**

In a positive learning experience, students are able to develop their mental model of the material through productive feedback and feelings of success. The instructor's role is to facilitate that experience by providing useful and timely feedback. The instructor must also be prepared to appropriately handle negative situations, such as grading concerns or student confusion.

Although feedback is a major focus of the instructor's role, other factors contribute toward student success and self-image. We can explore how much of an impact feedback and these other factors have on student experiences, and their potential for introducing or validating methods of instruction. We can also learn from students' negative experiences to improve their future experiences.

**What potential for model regression is possible, and how can that potential be mitigated?**

Kolb provides requirements of the students for successful implementation of the ELM. Students must be involved in the experience, reflect on it, conceptualize their model and act on their newfound understanding [42]. If we are to create a successful learning environment, we must be able to confront and encourage positive student behavior while being aware of common student biases, such as the belief that creating automated tests is wasted energy.

The three courses presented in Chapters 2, 3 and 4 provide insight into experiences involving introducing testing elements. The experiences in each course provide a unique scenario of the ELM. When combined, we learn the potential of testing in the classroom and methods for providing positive experiences with testing.

## 1.3 JUnit Testing Experiences in an Introductory Programming Course

Chapter 2 focuses on the JUnit Generator, a program that an instructor can use to create and run unit tests, then generate reports for students. Ultimately this tool enables instructors to provide more experience in the form of assignments or labs.

As discussed earlier, my second instructional position was in a data structures course. Drawing from my earlier lecturing position, I recognized the need for students to get applicable feedback. This is consistent with Kolb's discussion of feedback and would allow students to apply knowledge from both successes and struggles into future work. To accommodate this, we doubled the number of assignments in the course.

The increased assignments in the course helped students, but put an excessive burden on the grader. To compensate, tests were used to automatically execute the program with contrived data. If the program worked as expected, the output would be the same. This assisted somewhat, but if there were errors in the output, the grader would have to manually determine where the error was introduced. Adding to the problems were ambiguous instructions, errors in the hosting executable and other snags. Ultimately, the instructor or grader would provide inadequate feedback or feedback that was too far behind schedule to be of use.

Soon, we began to use unit testing to validate programs. First we used a locally developed unit testing system that directly interacted with the code under test. The architecture made modifying tests for new assignments time consuming and difficult, even with small changes. In addition, if one of the tests did not integrate with the tests easily, the grader and instructor would be required to spend a large amount of time correcting the error before returning the reports to students. Later we developed the JUG system, which would generate unit tests from simple test statements, then compile them with student programs to create reports [15]. Support was also added to provide time analysis, which allowed additional feedback for data structures.

When moved to an accelerated introductory programming course, I recognized the need to provide many cycles of experience in a limited time. The use of the JUG grading tool decreased evaluation time and allowed faster grade return. The most recent run of the course included 21 programming labs and assignments in a 15 week course. The coding lab was a two-hour time period with a teaching assistant, and the JUG system was used to create and send reports approximately every 15 minutes. This allowed students to have up to 5 experiences with instructor feedback on a single program.

This introductory course focused on a different aspect of the experiential learning model, and provided insight into the research questions. The course focused on fast cycles of programming assignments (CE) and feedback (RO). The advent of the JUG tool allowed the instructor to provide the feedback with less addition to the course than might be expected for a large number of assignments. Although difficult for students, the fast cycles meant they had their model of computer programming expanded quickly and consistently with no noticable regression.

## 1.4 Integrating Roles into a Classroom Experience

Chapter 3, *Combining Distinct Graduate and Undergraduate HCI Courses: An Experiential and Interactive Approach* describes a classroom with two courses, integrating two different professional roles into one course [13]. The original intention of the course was to provide a professional experience for both the graduate student as a usability expert and the undergraduate as a developer. In the first iteration of the course, the instructor acted as the customer, though later versions of the course included additional external roles.

The experiences presented in the combined course were derived from the original course, as described by Pastel [61]. This layout of the course also follows a similar scheme to the ELM process. Both Pastel [61] and Kolb [42] compare their process to the Scientific Method [59]. Pastel uses the Scientific Method as a basis for the undergraduate course. Similarly, in Figure 1.2, Kolb shows the similarities between the ELM and other intellectual processes. Pastel's cycle of proposal, test development, evaluation and analysis correspond to Kolb's Inquiry/Research cycle (outermost).

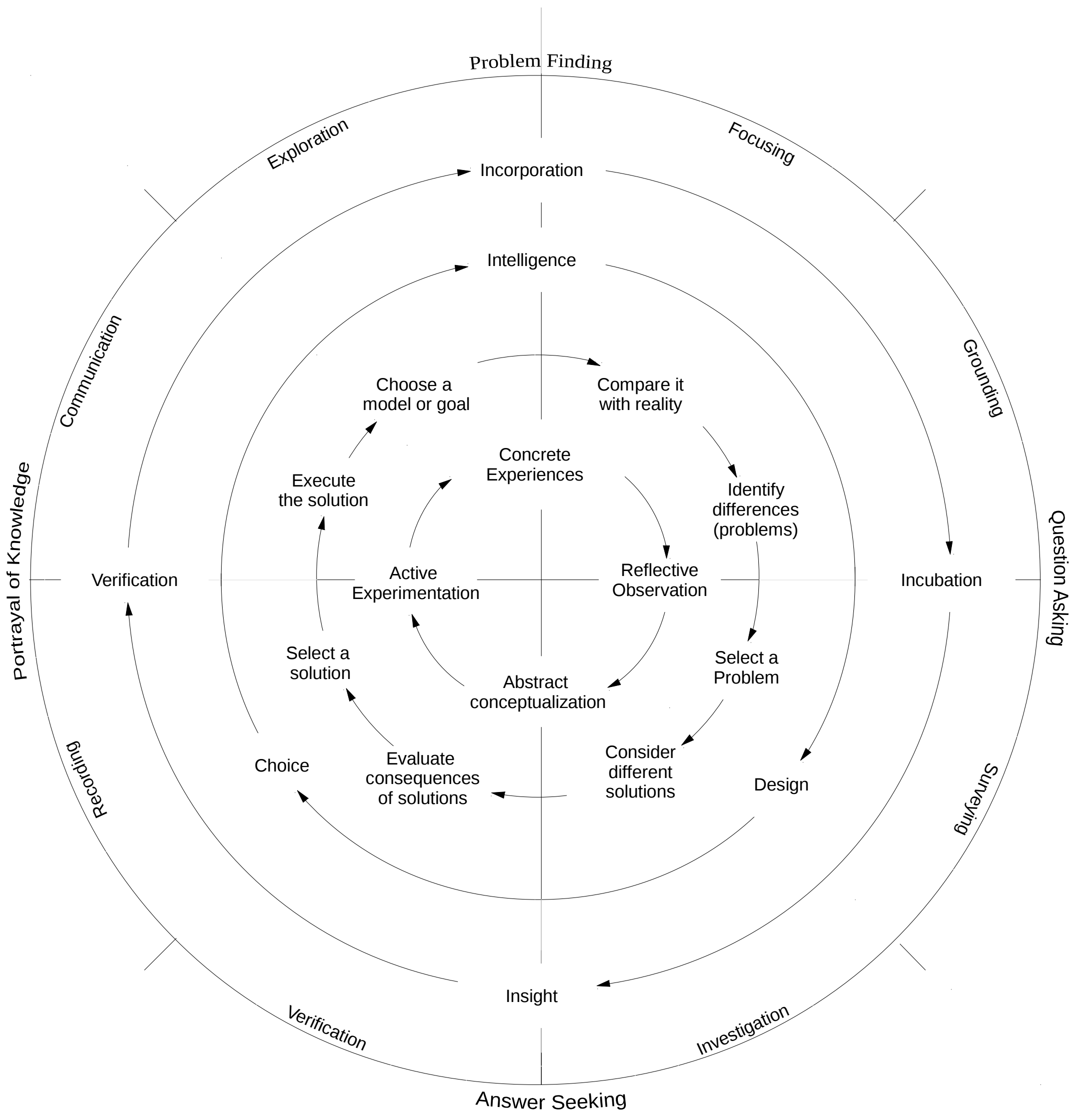


**Figure 1.2:** Kolb's Experiential Model expanded to include other process models: (from outside) Inquiry/Research [Kolb], Creativity [Wallas], Decision Making [Simon], Problem Solving [Pounds], Learning [Kolb].

One of the primary experiences when developing and running this course was the management of a multi-course classroom. The lecture material had to be applicable to both graduate and undergraduate students, and lecture time needed to be set aside for interactions between the groups. In addition, graduate students were expected to research and present new research into HCI, providing them valuable research experience. For the undergraduate students, feedback was an important addition to the course. The graduate students provided feedback throughout the course, independent of the instructor, giving undergraduates a broader perspective of the usability of their design and product.

Since the publication of this paper, additional elements have been added to the course. The original course product was for a fictional hardware product with intentional restraints. With the advent of Google's Android Software Development Kit (Android SDK) [28], students in the course were able to freely develop for a portable device. Next, as part of a collaborative Citizen Science project, real customers were added to the course, so that students were designing to a specific need, rather than their own project [64]. The most recent addition to the course is a group of students participating in a Usability Instructions and Writing course, part of the Humanities department at Michigan Tech [58].

This multi-role HCI course provides insight into the main research questions concerning experiential learning in computer science. The focus of the course was validating high level concepts of Human-Computer Interaction (AC) and actively researching those concepts (AE). Ultimately, the course required a significant number of additional steps on the part of the instructor to ensure a positive experience. Finally, the instructor and graduate students provided feedback to undergraduates, but based on survey results, there was regression in some areas that could be mitigated.

## 1.5 Adding Unit Testing to an HCI Course

Chapter 4 provides insight into the attempted introduction of unit testing into an existing project course to improve quality [14]. It continues from the research done in Chapter 3 and the Citizen Science project [64]. Before the Citizen Science project, the HCI course focused strictly on usability, allowing students to participate in the final usability testing portion with an incomplete prototype. However, with the addition of a customer, there was a desire to encourage functionality in the projects. The problem was that of motivation - if students are graded based on specific usability elements, what compels them to provide a fully functioning product?

We decided to include an additional experience in the course: Unit Testing. The Android SDK had compatibility for unit testing, and tutorials for using it [28]. Students could add unit tests to their project, and from those tests, ensure functionality. Unit tests also added an additional experiential learning potential. Writing unit tests was not necessarily part of a required course for these students, so requiring students to write unit tests, meant they could learn more about unit testing.

In the HCI course discussed in Chapter 3, one of the major components was the Usability Testing at the end of the semester. The concept of testing was to provide a culmination of the project, but also to give students feedback on their own usability development. The addition of unit tests seem like a natural addition to the project, and allowed students to present their project's functionality with its usability.

To help prepare students for unit testing, they were required to complete one of the Android SDK tutorials. Later, a lecture discussing unit testing provided practical applications of unit testing and the opportunity to ask questions. When their project moved to the development stage, each group was asked to unit test one element of their project. The graduate student evaluating the usability would provide bug report forms to the group, which would be filled out during usability testing if any functionality problems were encountered. In previous years, these problems would have been fixed quickly and testing would continue, but no record existed. The intention of the record would emphasize the importance of functionality. Finally, the groups took a survey about unit testing, and were interviewed about the unit tests they did.

This course answered the research questions in an unexpected way. Our own experience, observation and abstraction led us to believe that adding Unit Testing experiences to an already experienced based course would be natural. Instead, we found that the students' internal model of unit testing was much more resistant than we had thought. The focus of the research was on observation (RO) and conceptualization (AC)—students would see the benefits of unit testing and apply it to other situations. Unfortunately, it was implemented as an addition to an already project-heavy course. Although students anticipated being able to add unit testing to their code, most groups did not feel they had time to test. Based on our survey results, much more effort is needed to mitigate the regression.

## 1.6 Summary

While adding testing to various points in the Computer Science curriculum at Michigan Technological University, we provided experiences to help the students learn testing practices and benefits. Our hope is that testing practices be incorporated into more curricula, and the ideas of experience based learning can be used to introduce other Computer Science concepts.

The next three chapters are derived from independent publications. However, when examined through the lens of introducing test experiences into a classroom, they all reflect Aristotle's *techne* learning and Kolb's ELM. Additional information has been added to each of the chapters to expand on the research presented and to tie the work to the central thesis of test experience. In addition, a discussion section has been added to Chapters 2 and 3 to relate the original elements of those papers back to our original research questions.

# Chapter 2

# JUnit Testing Experiences in an Introductory Programming Course

The material for this chapter was adapted from *JUG: A JUnit Generation, Time Complexity Analysis and Reporting Tool to Streamline Grading* [15].

## 2.1 Introduction

According to the experiential learning model[42], feedback is an important element of the experience cycle. In addition, to avoid model regression, that feedback must be quick, detailed and consistent. The lack of appropriate feedback, especially in early programming courses, could lead to incorrect mental models of programming. Therefore, an instructor's role is to provide appropriate feedback in a timely manner.

One decision that introductory programming instructors must make is balancing the number of programming exercises with the amount of time available for preparation and grading. Additional practice of new skills and ideas helps developing programmers, but can increase the amount of work an instructor must do. Although automated systems can decrease the time spent grading an individual student, the time spent setting up the automated system and managing unusual cases still makes fewer assignments a more appealing option.

There are fundamental differences between a classroom grading system and professional software testing. In a professional environment, tests are meant to identify and prevent problems with the software. Often those tests are designed with edge cases and provided data structures in mind. In a classroom environment, the instructor must assign weight to different aspects of the program, and in certain courses (such as a Data Structures course), students are expected to write those objects that industry takes for granted. In addition, an industry project may be developing the same code for months, without much need to write new tests. An instructor must prepare tests for each assignment, where a single assignment is only used once during a week. Finally, reports given to a grader or student must include more detailed reference to how the test was executed, since the student would not have direct access to the test.

The two major goals for the JUnit Generation (JUG) system are: 1) providing strong supplemental feedback to improve student learning, and 2) ease of use for the assignment developer and grader. The JUG system combines automated tests, evaluation and reporting to fulfill those goals.

Our first goal, course improvement through fast, useful feedback, is attained by providing concise but clear test descriptions and results, and by including important supplemental information into the automated process, such as time complexity analysis. We measure the success of this method by examining the types of assignments and grading techniques used as our automated grading system evolved. In addition, we have interviewed the instructors and graders, and surveyed students to gain more insight into the success and future work of the system.

To create an easy system for instructors developing assignments and graders, we developed an environment where a single, simple test requires only a single, simple line of code. The system is flexible enough to incorporate multi-line tests, and can assign point weight to different test groups. A report generator can also produce a graded or partially graded report for the grader.

## 2.2 Background

Automated testing is not new, in fact, one of the first publications using tests in the classroom is from 1960 [32], with a goal of saving the then precious machine operation time. As additional resources have been developed, and resources available to instructors improve, automated instruction has increased potential. Although many automated grading systems already exist [35], the fact that they are still being developed indicates the strong need for adaptive systems.

### 2.2.1 Previous work on Automated Grading

Recently, Ihantola et al. reviewed a number of automated grading programs with associated publications between 2005 and 2010. Their assessment indicated that programs should include stronger sandbox/security systems, more public distribution methods, and combine automatic feedback with manual feedback [35]. We have attempted to follow these guidelines through our goals by using JUnit, publishing JUG (see Section 2.4.4), and creating reports ready for the grader to examine.

Venables and Haywood stress the importance of feedback when developing their web-based grading tool, *submit*. They used a comparative output providing the difference between the expected and provided results [71]. We also adopt this technique, although JUnit provides some utility for more succinct comparisons (see Section 2.4.1). Finally, Kay et al. provide a set of important ideas regarding the philosophy behind grading [41]. We address many of these ideas in Section 2.3.

### 2.2.2 Unit Testing and Test Driven Development

One of the common testing methodologies is unit testing. Unit testing is a testing approach with the following properties: Simplicity, Independence, and Documentation[6]. Tests focus on a unit, or single piece, of the overall functionality within the program, ensuring that they work independently, for smoother integration. The unit can vary widely in size, from a single function, collective behavior of methods within a class, or a group of classes which work together. By being simple and independent, the result of a unit test should be consistent for repeatability. Unit testing is often automated, and a benefit of creating automated tests is that the purposeful thought of creating the tests causes students to avoid and discover errors quickly.

The **Test-Driven Development** (TDD) methodology is based on the mantra: “write a test; make it run; and make it right” [25]. In other words, tests are a fundamental step in agile programming. TDD practices can demonstrate improvement in both productivity and quality of the final product [9]. Erdogmus et al. demonstrate that the test-first approach, when used with incremental development, promotes the creation of more tests, thereby providing stronger code [24]. By focusing the JUG unit tests in a TDD style, especially in Data Structures, students experience first-hand the benefits of making tests for their own projects. In addition, since JUG is publicly available, students and other developers can use it to easily create and maintain tests.

### 2.2.3 JUnit

JUnit [7] was developed as a tool for test driven development, and has been expanded to serve in different ways [50, 53, 65]. JUnit developers use the terms *test cases* and *test suites* to represent the organization of tests. Often, these tests are organized by what aspects of a class they test. This is similar to a point distribution organization in any graded assignment, so JUG exploits this similarity. JUnit also has the benefit of running the tests as separate threads, avoiding complications from errors propagating into the testing environment.

### 2.2.4 Generative Programming

Generative programming techniques provide the means to combine domain specific abstract concepts and template structures to create code in a general purpose language, such as C++ or Java. Czarnecki and Eisenecker standardized generative practices in 2000, focusing on constructive techniques to use abstract terms to describe a system, without manually creating each part [19]. Since then, a number of domains have begun using generative programming, including embedded systems, scientific applications and user interface generation [3, 4, 20, 29, 31, 68]. While our automated testing evolved, we noticed a few recurring patterns in the test styles and overhead, making templating techniques the natural choice for generating tests. This evolved into JUG as a domain-specific language for automated testing.

## 2.3 Philosophy of JUG

Kay et al. provide a set of issues that an automated grading system should address, followed by a breakdown of the aspects of an automated grading system [41]. Although we diverge on some of the thoughts behind their proposed ideas, it serves as a good baseline for discussing the decisions behind JUG.

### 2.3.1 General Issues with Automated Grading

The general issues Kay emphasizes are security from tampering, scalability to large class sizes, flexibility to varying assignments, and robustness toward bugs. Many of these issues have been solved through JUnit, since the unit tests are run in separate processes, can handle many different types of code, and are usually robust in dealing with errors. However, to resist tampering, JUG uses Java's Security Manager class to avoid improper student calls, such as `System.exit`, which would end the entire testing process.

Our system scales by separating the test generation from report generation. The tests for all students are the same; therefore, generating the JUnit tests once is sufficient for the entire class. When compiling and running the tests, however, each student is graded independently. We use `BASH` scripts to direct the processes, avoiding processor scalability issues by operating on one student at a time. This process is completely automated and does not requiring human intervention that can slow the process down for large classes. Afterward, the human grader can examine and finish grading the stored reports. To add to the robustness, timeouts can be set so that a student with an infinite loop can still receive a readable report, and the system can grade the other students.

### 2.3.2 The Automated Process

#### 2.3.2.1 Submission

The first suggestion from Kay et al. is that a submission system should allow for multiple source files and ancillary files [41]. With object-oriented design integrated into early programming courses, this is a necessity. Fortunately, our department has a submission procedure established. However, with the ability to export projects through Eclipse, create branches using version software, and additional means for students to provide code to the instructor, this aspect is left to `BASH` scripts. The JUG framework includes appropriate `BASH` scripts to manage code for compilation and testing (see Section 2.4.4).

#### 2.3.2.2 On-Submit Feedback

In addition to submission, Kay et al. suggest that feedback be provided immediately after each submission, such as compilation information and results of tests against published cases [41]. Instead, we consciously do not permit students to see this type of feedback before the due date. This prevents students from solving the problems by trial and error or using the grading program as a debugging tool, as experienced by Malmi et al. [52]. The experience of compiling, testing and debugging is part of the programming process, and once a student begins using the grading system as a debugging tool, they lose the experience of testing their own code.

#### 2.3.2.3 Two-Phase Feedback

To provide students with the benefits of near-immediate knowledge of the results, we use JUG in a two-phase submission process. At the due date, reports are generated by JUG, but not evaluated by a human grader. Those ungraded reports are emailed to students, giving them the ability to see compilation errors and failed tests. A few days later, they may choose to resubmit a fixed program for a small penalty. JUG creates reports for the resubmits, then the human grader evaluates the programs. This means that the human grader only looks at each student's grade once, but students get the benefit of immediate feedback and the opportunity for improvement.

#### 2.3.2.4 Test Components

The obvious role of test components is to determine whether the student's code works as required. However, the more subtle aspect of this role is the pedagogical feedback provided. To assist in improving student understanding, we want students to be able to recreate the test case and know exactly what is expected. For this, JUG uses annotated comments, and includes more descriptive feedback.

#### 2.3.2.5 Scoring

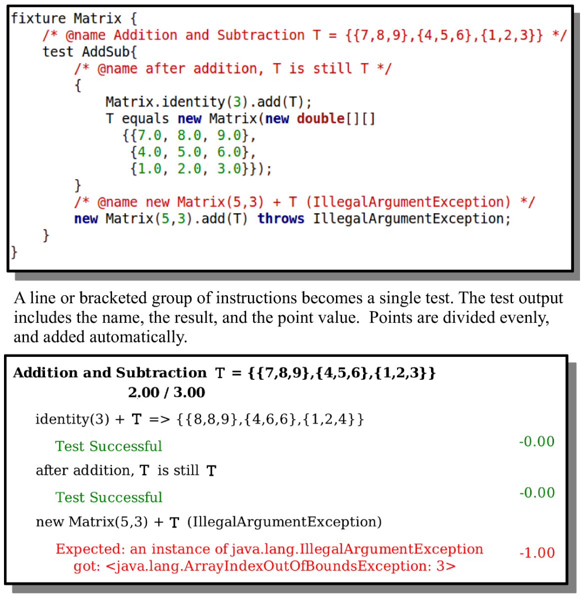
Kay et al. suggest that grades or point assignments not be calculated, but left for a human grader. Their reasoning includes adjudication of common mistakes, such as spelling and whitespace in the results. Although these common errors are important to adjust for, we have found that presenting students with a low score due to minor errors emphasizes the importance of attention to detail required for programming. In addition, with the regrade policy described above, a quick fix of these minor errors creates a minimal penalty. On the other hand, including a grade calculation can reduce the grader's time spent adding and calculating grades, which can cut grading time significantly.

#### 2.3.2.6 Plagiarism

Kay emphasizes the importance of automated plagiarism detection, and we agree with that assessment. Our department uses the freely available plagiarism detection system, Moss [1], to evaluate student code. Through some simple organizational scripts, Moss has allowed us to compare students in the same course with each other and those from previous semesters.

## 2.4 The JUG System

JUnit Generator, or JUG, consists of two parts: a test generator, and a report description. By mixing of natural language and Java code, JUG's input syntax can provide sets of simple JUnit tests, while maintaining an overview of all tests.



```
fixture Matrix {
    /* @name Addition and Subtraction T = {{7,8,9},{4,5,6},{1,2,3}} */
    test AddSub{
        /* @name after addition, T is still T */
        {
            Matrix.identity(3).add(T);
            T equals new Matrix(new double[][]
              {{7.0, 8.0, 9.0},
               {4.0, 5.0, 6.0},
               {1.0, 2.0, 3.0}});
        }
        /* @name new Matrix(5,3) + T (IllegalArgumentException) */
        new Matrix(5,3).add(T) throws IllegalArgumentException;
    }
}
```

A line or bracketed group of instructions becomes a single test. The test output includes the name, the result, and the point value. Points are divided evenly, and added automatically.

```
Addition and Subtraction T = {{7,8,9},{4,5,6},{1,2,3}}
              2.00 / 3.00
   identity(3) + T => {{8,8,9},{4,6,6},{1,2,4}}
      Test Successful                                        -0.00
   after addition, T is still T
      Test Successful                                        -0.00
   new Matrix(5,3) + T (IllegalArgumentException)
      Expected: an instance of java.lang.IllegalArgumentException   -1.00
        got: <java.lang.ArrayIndexOutOfBoundsException: 3>
```

**Figure 2.1:** Test Generator File and Report Samples

### 2.4.1 JUnit Generation - The JUG File

The JUG system uses templates to generate JUnit tests. Our objective for the generator was to have one line of JUG for each test. Although this meets our secondary goal of simplifying the grading process, tests must also be able to provide adequate feedback to students so that they can recreate the tests and improve their programs.

For each simple test, we use the form [`code keyword code;`] to represent a JUnit test statement. Some of the common keywords include `equals`, `arrayequals`, `isinstanceof` and `throws`. These terms use templates to expand to a single JUnit test statement within a test method. Since this type of statement indicates both the call and the result, it can be easily reproduced, and clearly represents the test being called. This test statement is used in the report to describe the test to the student.

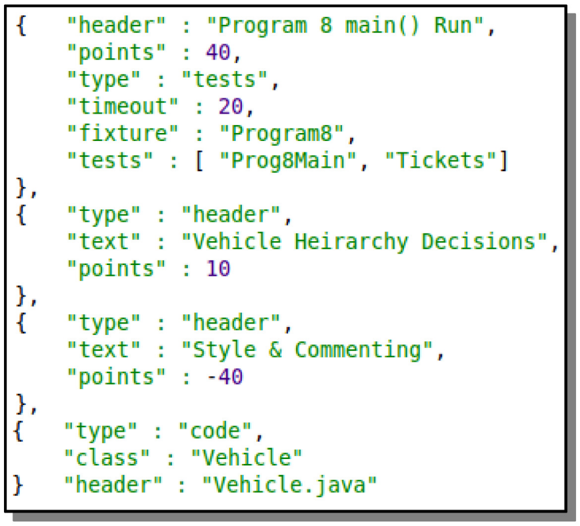
Sometimes more complex tests are necessary, however. Therefore, JUG allows a block to be created representing a single test, where the last line of the block is the test statement. Any valid Java code can be placed within the block, allowing any kind of set up to be made prior to the test. In this case, the test statement may not be adequate, so a simple javadoc-style annotation can be added as a descriptor for the test, allowing the test developer to describe the test. See Figure 2.1.

### 2.4.2 JUG/JUnit Reports - The JUR File

The reports are defined using JavaScript Object Notation (JSON), which uses simple formatting rules to divide data into generic lists and maps [18]. We chose this notation over other standards, such as XML and HTML, because the notation is straightforward and intentionally human-readable. In addition, by using a standard convention, code highlighting is already available.

The report format is based on various section types. The most complex section type is the test section, allowing the instructor to set the number of points for the section, or for individual test suites. Test suites are referenced by name to the JUG file. Additional sections provide annotation tables, plots and source code (see Figure 2.2, Figure 2.3 and Section 2.4.3).

Generating the report uses an abstract report generation class, with abstract methods for "printing" the various components of a report (i.e. tests, headers, plots, etc). Through these methods, a subclass generates an XHTML document which can be posted online, emailed to students, printed, or converted to a pdf. Currently, we convert the files to pdfs, however, the system is flexible enough that one could generate reports in many formats.



```
{   "header" : "Program 8 main() Run",
    "points" : 40,
    "type" : "tests",
    "timeout" : 20,
    "fixture" : "Program8",
    "tests" : [ "Prog8Main", "Tickets"]
},
{   "type" : "header",
    "text" : "Vehicle Heirarchy Decisions",
    "points" : 10
},
{   "type" : "header",
    "text" : "Style & Commenting",
    "points" : -40
},
{   "type" : "code",
    "class" : "Vehicle"
}   "header" : "Vehicle.java"
```

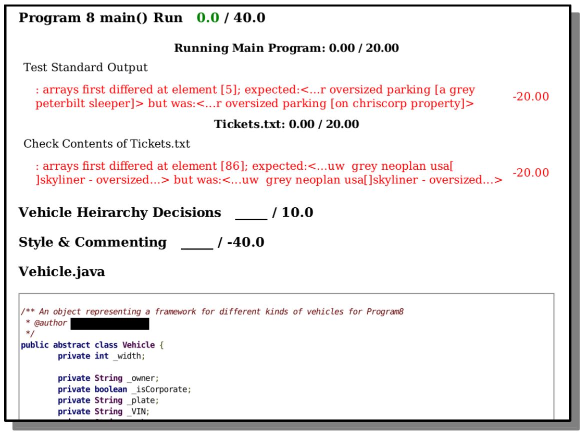
**Figure 2.2:** JUnit Report File (JUR)

### 2.4.3 Tables and Data Plots

Another objective for improving supplemental feedback is providing students with real analytic data concerning their programs. Therefore, the JUG system supports programmatic plot generation and annotation tables. See Figure 2.4.

Instructors can generate tables based on custom annotations. A reference to the class, method and annotation type allows instructors to acquire meta-data from the students, and print that information to the report. In our use of JUG with Data Structures (see Section 2.5.1), we used these annotations to require students to include time and space complexity information concerning their data structures.

For a data plot, the instructor includes code for two Java methods which will generate and return data arrays for the plot axes. They can include calls to known student methods, and time those methods or reference results. In our use, we included time complexity plots by running methods with various inputs and data structure sizes.



```
Program 8 main() Run   0.0 / 40.0

                    Running Main Program: 0.00 / 20.00
 Test Standard Output
   : arrays first differed at element [5]; expected:<...r oversized parking [a grey
   peterbilt sleeper]> but was:<...r oversized parking [on chriscorp property]>      -20.00
                         Tickets.txt: 0.00 / 20.00
 Check Contents of Tickets.txt
   : arrays first differed at element [86]; expected:<...uw  grey neoplan usa[
   ]skyliner - oversized...> but was:<...uw  grey neoplan usa[]skyliner - oversized...>  -20.00

Vehicle Heirarchy Decisions   ____ / 10.0

Style & Commenting   ____ / -40.0

Vehicle.java
```

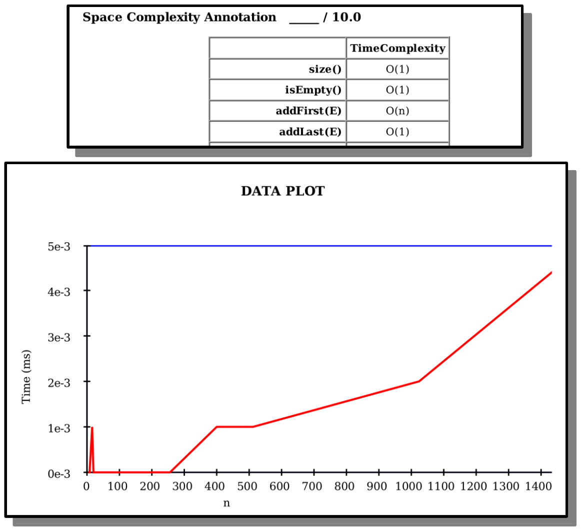
```java
/** An object representing a framework for different kinds of vehicles for Program8
 * @author [redacted]
 */
public abstract class Vehicle {
	private int _width;

	private String _owner;
	private boolean _isCorporate;
	private String _plate;
	private String _VIN;
```

**Figure 2.3:** JUnit Report Output

## 2.4.4 Using the JUG System

The JUG system uses a series of scripts to aid in use of the test generator and report generator. First, the instructor must create a solution, a test file (JUG) and a report file (JUR). Next, the instructor modifies a configuration file for the `BASH` scripts. Finally, a



Space Complexity Annotation ____ / 10.0

| | TimeComplexity |
|---|---|
| size() | O(1) |
| isEmpty() | O(1) |
| addFirst(E) | O(n) |
| addLast(E) | O(1) |

**Figure 2.4:** A Time Complexity Table and Graph

grader can run scripts to gather student submissions, generate JUnit tests, create the reports, and email the reports to students. If any students need to be tested again, scripts include options to restrict a run to individual students.

## 2.5 Evaluation

Our primary goal was to use automated testing to provide quick, consistent and detailed feedback to students, to promote correct mental models of computer programming. We used automated grading to provide that feedback in a way that allows faculty and graders to best use their limited time. We also wanted to introduce testing practices, such as unit testing and test driven development. Our two case studies allow us to demonstrate that we met our primary and secondary goals.

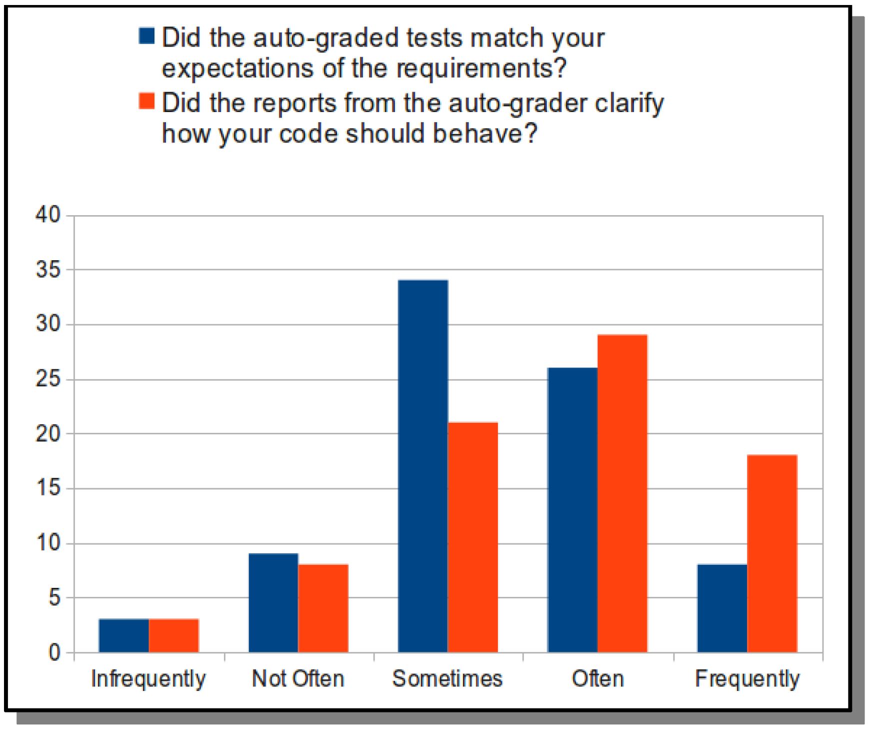


**Figure 2.5:** Student Survey Results

### 2.5.1 Case Study: Data Structures

Our first case study was a data structures course. Before automating the system, the course included five assignments. Four of the assignments required creating a data structure and an application of that data structure. The fifth assignment compared the time complexities of various sorting algorithms, requiring students to graph and report their results. Observations of student programs indicated that they would use incomplete data structures, meeting the minimum requirements for the application. In addition, the assignment provided little distinction between the data structure and the application, and often students had difficulty getting various components to work together correctly.

We introduced unit testing for the data structures component of the assignment, and generated reports for students. This divided each assignment into two separate parts, the data structure and the application, eventually becoming two distinct assignments. It also allowed us to include elements of time complexity analysis into the data structure assignments. The difficulty in automation was writing the unit tests and the report generator for each assignment. Often, the largest time costs were specifying the form of the report in code, and formatting the unit tests appropriately for unusual types of tests. In addition, our original version had points assigned per test, creating problems when tests were added or removed—many times the total points for an assignment was an unusual value. Automating these aspects led to the template based test generator, and a simplified report system that automatically divided test points.

Students responded well to the JUG reports, using them to discuss their expectations and grades. We conducted a survey of the students to determine the perceived impact, and attempted to infer the real impact. The survey questions are provided in Appendix A Two questions from the survey directly inquired about the JUG reports:

1. Did the auto-graded tests match your expectations of the requirements?
2. Did the preliminary reports from the auto-grader clarify how your code should behave?

The results of these two questions are shown in Figure 2.5.

We believe that these results indicate that students found that the reports helped them better understand the data structures and their applications after seeing specific tests run. For instance, a student may not understand why specific types of exceptions are thrown. Seeing a test cases throwing those exceptions can improve that understanding through the experience.

### 2.5.2 Case Study: Introductory Programming

The second case study incorporated the JUG system into an accelerated introductory programming course. The accelerated course covered material from two introductory courses in one semester. The course was presented in an algorithm-first approach, incorporating object-oriented design practices in the fourth week (of fifteen). Due to its accelerated nature, the course included 15 assignments, meaning that the turnaround time for a program was often less than a week. Because of this, we decided that resubmissions would only be allowed in special cases, but often there was no penalty associated with resubmission.

The results provided strong evidence that JUG improved the efficiency of the grading process. The assignments were all new, but tests were simple to generate and most reports could be copied from one assignment to the next with minimal changes. One major problem was testing extremely long outputs. One report printed two pages of “expected” output, followed by another two pages of slightly incorrect output. To alleviate this, JUG allows output to be split by line into an array. An example of a result like this is displayed in Figure 2.3—students were simply told that the array index represented the line number of the first error.

During the Fall 2012 iteration of the course, at my suggestion, a lab was added to increase the quantity of experiences in the course. To account for the change in structure, the original 14 homework assignments were split into 10 laboratory assignments and 11 homework assignments. During the laboratory assignments, students were provided feedback approximately every 15 minutes, and were able to work against their test results, much like Test Driven Development is done.

### 2.5.3 Results

The two case studies demonstrated that the feedback provided by the JUG was quick, consistent and detailed. Students were able to see their unit test results immediately after the assignment was due, and were provided the opportunity to see the results again after resubmissions. Based on the survey and introductory course laboratory exercises, students were able to better understand the expectations and requirements of the assignments. Finally, since the students understood that their programs would be strictly tested, they could write their own rudimentary tests. The resubmission and laboratory assignments demonstrated the concepts of unit testing and test driven design in a concrete experience.

#### 2.5.3.1 Emergent Results

As we used the grading tool, we discovered two aspects to the tool that improved the classroom experience. The first benefit was that of communication; because of the clarity of feedback, students were better prepared to discuss the elements of their program that were broken. The second benefit that the system provided was adaptive reusability; simple changes to a JUG definition file allowed the instructor to provide variations on their assignments between successive semesters.

In programming assignments, students will often make claims that the program "works" to an extent better than their grade reflects. Of course, this is a very subjective statement and often difficult to argue against. However, when the grade is presented as a series of rewards and penalties, students instead tend to argue against those penalties they believe are incorrect or inconsistent. This provides a moment where the instructor either a) finds an error in their requirements, tests or grading policy, or b) provides an instructional moment with the inquiring student. Both are positive situations which result in a satisfactory conclusion for both instructor and student.

One of the unexpected benefits we found using this system was the adaptability and reusability that the system provided. After writing only a few sets of tests, we found we could copy and paste sets of tests within an assignment, and quickly modify them to a new case. We could also take series of tests from one assignment to another and make minor modifications. Often, between one semester and another, we would make minor changes to the assignments and our solution. We could then use JUG to run the old test on the new solution, generating a report of which tests needed adjustment. The JUG and JUR files are now regular components of our assignments with the description and solution, and are easy to port from one semester to another.

## 2.6 Conclusions

The goals of the JUG system were improving student understanding and easing the grading process. Based on the evidence presented, we believe this was the case, and that the results of streamlining grading exceeded our expectations. Based on the case studies, students found that JUG helped their understanding of various aspects of programming, and we, as instructors, found the system easy to incorporate into our classroom, improving the efficiency of our work.

## 2.7 Discussion

The experiences and feedback provided to the students represented many successive iterations of the Experiential Learning Model. Each assignment used a concrete experience (CE) of development and testing to help students reflect (RO) as they build their mental models of basic programming concepts. The material was supplemented in lectures, providing the abstraction (AC) necessary to cement that model and prepare them for future concepts.

The grading process enabled by the JUG tool provides insight into our three central research questions. First, the increase of programming experiences with appropriate feedback is beneficial to providing correct mental models. Second, although an increase in work is apparent for an instructor, the necessity of automation frees faculty and graders from focusing on the mundane aspects of the assignments, so that they can focus on the important elements of the course. Third, an instructor must be conscious of students who fall behind, especially in a fast-paced course, since not participating in the experience (by not completing the assignment) prevents feedback, and ultimately retards the development of the proper mental model.

# Chapter 3

# Integrating Roles into a Classroom Experience

The material for this chapter was adapted from *Combining Distinct Graduate and Undergraduate HCI Courses: An Experiential and Interactive Approach* [13]. The graduate course was developed in 2008 and an iteration of the combined course has been run each year. The course has developed since this original publication, as described in Chapter 4.

## 3.1 Introduction

A combined course is one in which students enrolled in distinct courses gather in the same room, making insights on the same topics. When combining two courses, it is important to consider the reason they are being combined, and the purpose for each course. For these combined courses to be truly distinct and not cross-referenced courses, such as the multi-faceted course in [72], the students enrolled in the two courses should benefit in different ways through the development of distinct ideas and skills.

Frequently, graduate and undergraduate courses are combined by requiring the graduate students to write an additional term paper and make a presentation to the class. Combining the courses by requiring more work from the graduate students does provide higher standards and ensure a richer learning experience for all students. However, after taking one course, neither graduates nor undergraduates are inclined to enroll in the counterpart in subsequent semesters. Our standard for combining the graduate and undergraduate Human-Computer Interaction (HCI) courses was to ensure that the learning experience for both the graduate and undergraduate students was distinct enough that any student could benefit from, and would desire to take, both courses.

HCI is a particularly promising area for combining distinct graduate and undergraduate courses because it is a broad discipline requiring many skills to implement a successful user interface (UI). HCI is the convergence of at least three disciplines: software engineering, design, and cognitive science [55]. In previous iterations of our undergraduate course, the emphasis was on UI design [61]. However, an interface designer should not only be a skilled programmer, talented in visual and interactive design, but also be able to conduct and evaluate how users will interpret their design. The earlier course did not provide adequate experience evaluating and testing their UIs, and consequently, the HCI students were convinced that their designs were satisfactory despite key usability concerns.

The skills required to implement a successful UI can be divided into two parts:

1. UI design and implementation
2. Evaluation, usability testing, and analysis

However, if a group performs evaluation, testing and analysis on their own project, many of the key usability concerns which were ignored through the design would also be ignored through testing. To have students both develop their own prototype and evaluate others would require a very large time commitment to the course. Although time might permit creating a low-fidelity prototype for this purpose, current research suggests that the benefit of testing high-fidelity prototypes provides more compelling evaluations [48, 76].

The division between developer and evaluator is a natural one to follow when dividing the graduate and undergraduate courses. The undergraduate students at our university are especially proud of their programming skills, and they are eager to design and implement their own unique UI. Furthermore, graduate students are expected to have better analytical skills then undergraduate students. Consequently, students enrolled in the undergraduate course could design and implement a UI, while students enrolled in the graduate course evaluate and test those UIs. This provides an additional level of cooperative learning which is rarely seen in the classroom [26].

Besides ensuring that the combined course provides a distinct and meaningful learning experience for all enrolled students, we had other goals for the combined course that reflected the scheduling and instructor's workload constraints. One goal was to do external user testing on a high-fidelity prototype. The undergraduate groups should have sufficient time to produce high-fidelity UI prototypes suitable for usability testing. Meanwhile, graduate students should have sufficient time to conduct the usability test and analyze the test results before the end of the semester. In addition, undergraduate groups should receive feedback on their design from several graduate students and not feel dominated by any single graduate student. Finally, a graduate-level course should provide the graduate students the opportunity to study current HCI research topics.

To provide appropriate experiences, we used Kolb's Experiential Learning Model (ELM) to introduce two combined, yet distinct, HCI courses. In the undergraduate course, students gain the experience of designer, while in the graduate course, students gain the experience of evaluator. Through these courses, we can emphasize the importance of design, software engineering, and cognitive evaluation in creative projects. Through the course, we emphasize the benefits of the interaction between graduate and undergraduate students, as well as the project development by the undergraduate students and evaluation process by the graduate students.

## 3.2 An Experience Based HCI Course

UI design is plagued by poor attitudes of inexperienced programmers, whose mental models of an intuitive, usable interface is ripe with problems obvious to HCI experts. To improve their mental model, we provide constructive feedback through analysis and user testing. We incorporate that testing through three iterations of the ELM, roughly corresponding to the three major sections of the course.

We introduced a graduate level course as a means of providing real, experimental evidence for students learning interface design in the undergraduate course that was already offered. The graduate course was primarily an evaluation course, where the students use projects from the undergraduate course to practice analysis of design proposals, heuristic evaluation of low-fidelity prototypes, and experimental study of high-fidelity prototypes. This gave the designers—the undergraduate students—meaningful usability feedback on a working project. In turn, the graduate students' concrete experience (CE) was that of writing reports detailing their findings, and the feedback was the student and instructor responses to those reports.

### 3.2.1 Course Descriptions

The undergraduate HCI course was a group project course that had been run in previous semesters [61]. Each group designs and implements a high-fidelity prototype application for a “Tiny Digital Assistant” (TDA), a very small device with only five hardware buttons, one of which is the power button, and limited screen space (320 x 240 pixels). Students worked through an iterative design process, including an initial proposal, a low-fidelity prototype, and a high-fidelity prototype.

The corresponding graduate HCI course was an experiential course. Graduate students took on the role of “expert evaluators”—usability experts who push groups to focus on user-centered design—ultimately testing the final prototype with sample users, and analyzing the results. The graduate students also explored current research areas in HCI, presenting their explorations to all students, including the undergraduate groups. Their exploration culminated in a short (5-10 page) written assignment on the subject. See the course website [63] for a complete schedule and detailed description of the assignments.

### 3.2.2 Student Interactions

Many of the assignments served two purposes in the course. They were used as graded material and as the method of communication between the graduate students and the

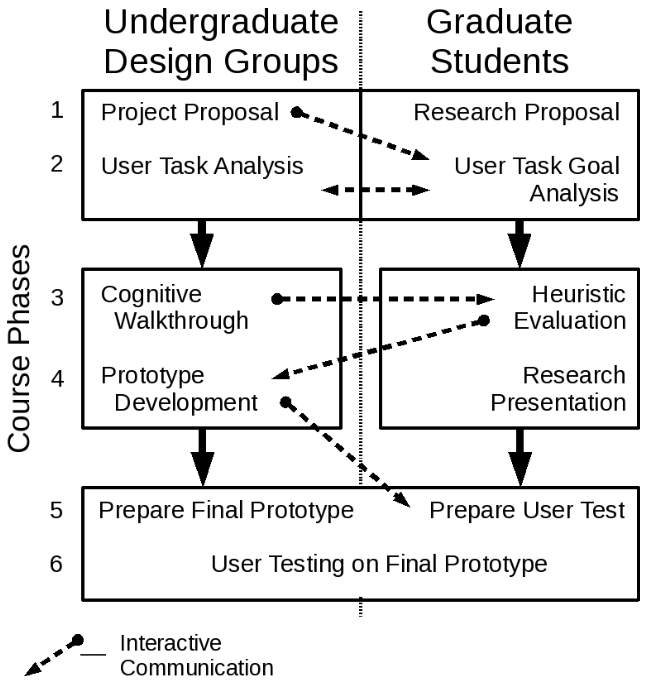


**Figure 3.1:** Interactions between undergraduate groups and graduate students throughout the course.

undergraduate groups. Both the groups and the graduates created websites to publish their documentation for others to review. This allowed everyone immediate and complete access to all of the documentation for each project.

The course was divided into three major sections, which are further divided into phases, as shown in Fig. 3.1. During each section, graduates worked with a different undergraduate group, reporting their findings back to that group. The undergraduates posted their design documents and the graduate student's feedback publicly, which allowed all groups to learn from each others' projects. As the class went on, the undergraduates could better understand each of the graduates' unique perspectives, and use those to develop their projects. Overall, this provides some consistency while still offering varied viewpoints.

#### 3.2.2.1 Initial Phases

In the initial phases of the course, the graduates and undergraduates had separate, but nearly identical functions within the class. The instructor lectured on fundamentals of HCI, while the students prepared for their distinct roles. First, each group and graduate student was required to create a proposal and a website (Fig. 3.1, Phase 1). This gave them an idea of what they should be preparing for the course—the undergraduate's group project and the graduate's research presentation and paper.

The undergraduate groups were asked to think about the primary users and the tasks they would undertake, to create a User–Task Analysis (Fig. 3.1, Phase 2). This helped the undergraduate groups focus on user-centered design for the remainder of the project. Meanwhile, the graduate students read one group's proposal, and met with that group to discuss the project and write up a User-Task-Goal Analysis (UTGA). Their function was to decide who the primary, secondary and tertiary users were, what goals they would have, and tasks necessary to achieve their goals. Then, those reports were provided as communication between the graduate students and their assigned group.

In terms of Kolb's ELM, the User-Task-Goal Analysis assignment was the graduate student's first concrete experience (CE) in evaluation. The graduates met with the undergraduate groups to discuss the analysis, allowing Reflective Observation (RO) of their analysis with their group's User-Task Analysis assignment. The graduate students were then switched to a different group, meaning they were encouraged to abstract (Abstract Conceptualization, AC) the knowledge of the first experience into their later evaluation (Active Experimentation, AE).

#### 3.2.2.2 Intermediate Phases

During the intermediate phases of the course, the graduate students worked independently of the undergraduate groups. There were only a few lectures by the instructor, since student presentations comprised most of the classroom time. The undergraduate students created a low-fidelity prototype (many of the groups created paper-prototypes), and presented this as a cognitive walkthrough to the class (Fig. 3.1, Phase 3). Each graduate student was assigned to two groups, evaluating each prototype using basic heuristics [60].

After the graduate students provided their evaluations to the undergraduate groups, the undergraduates worked at developing their prototype outside of class. Meanwhile, the graduate students developed their research topics into presentations for the class (Fig. 3.1, Phase 4). Though these presentations did not directly relate to the projects, it did provide a means for potential graduate students to learn more about the current research state of HCI, and specific areas they might be interested in researching.

In this ELM cycle, the graduate students' concrete experience (CE) was to evaluate the design and determine a set of heuristics that would be appropriate to evaluate. For the undergraduate students, this phase was an important cycle of the ELM. The low-fidelity prototype was their first concrete experience (CE) of design. The graduate students Heuristic Evaluation provided a foundation for reflection (RO). Since the next step was to begin implementation, they had to apply the feedback from the heuristic evaluation to the rest of their design (AC and AE).

#### 3.2.2.3 Final Phases

As the course was coming to a close, the instructor returned with additional lectures on advanced HCI topics—those that were not presented as part of the graduate's research topics. Outside of class, the undergraduates spent time developing their high-fidelity prototype, while the graduates prepared for the usability testing (Fig. 3.1, Phase 5). Since the graduates and undergraduates were preparing concurrently, the graduates could get updates on the status of the prototype. They could also offer suggestions to the undergraduates about providing more depth in certain aspects of the high-fidelity prototype for testing. Undergraduate groups would then know which other aspects could be left either incomplete or as a shallow prototype. Many of the graduate students also prepared monitoring programs which would run concurrently with the prototype, allowing them the opportunity to analyze their tests more closely.

During the final testing phase (Fig. 3.1, Phase 6), the interaction between the graduates and undergraduates culminated. Students from lower level CS courses volunteered to

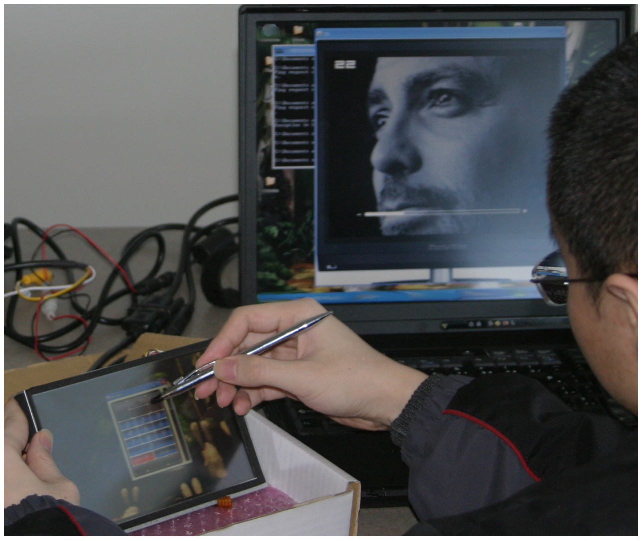


**Figure 3.2:** A user working with the Mini-Mote.

participate for a small grade incentive. Graduate students acted as the facilitators for the user testing, while the group members recorded data and monitored their prototypes for errors. This phase fulfilled many of the positive goals for the course to an extent beyond our expectation.

The final ELM cycle included the undergraduate students implementing their high-fidelity prototype, and the graduate students implementing a set of user tests (CE). The feedback for the cycle was the User Testing, but we wanted to be sure that the ELM cycle completed, so the final report and exit interviews allowed us the opportunity to ensure that students reflected (RO) and conceptualized (AC) the knowledge gained through user testing.

## 3.3 Example Projects

The 2008 iteration of the course consisted of 30 undergraduates broken into seven small groups, and seven graduates working individually. All of the undergraduates were Juniors or Seniors in the CS program. They had also completed a prerequisite of Team Software Project, a group-based software design and quality assurance course. Six of the seven graduate students were in the CS graduate program, with the exception from the Electrical Engineering program. Two of the CS graduate students had taken the undergraduate course before it was combined with the graduate course.

All of the groups successfully created working prototypes, which were then tested for usability with cooperation from students in introductory computer science courses. Two groups included a tactile aspect in their final prototype, allowing the user to test some of the physical constraints of using the device. The wisdom imparted by user testing was not limited to these tactile interfaces, though. In a third project, which was completely simulated on a desktop computer, students studied the difficulties inherent in designing expected functionality in a novel addition to a well-known device.

### 3.3.1 Mini-Mote

One project with a physical aspect was Mini-Mote, a universal remote-control application with a dynamic touchscreen interface. The Mini-Mote had built in controls for Televisions, DVD-players, and VCRs. The intended application is to be customizable for any device which accepts remote signals. Since the screen size was very small, and was intended for a human hand, the group developing Mini-Mote limited their menus to four items, one in each quadrant of the device.

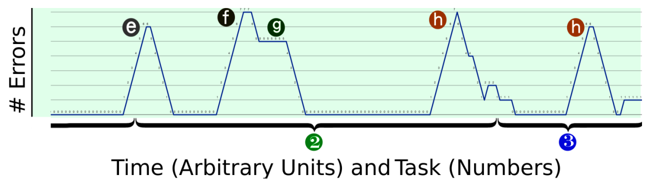


**Figure 3.3:** Click errors for one of Mini-Mote's users.

In user testing, the group created a prototype with a touchscreen (10in, 800x400 resolution) attached as a second monitor to a laptop. As can be seen in Fig. 3.2, the user controlled the "television" with a touchscreen remote control. They were asked to perform a series of tasks, such as turning on the TV, changing the channel, even adding a new device. Meanwhile, a background process counts the users clicks, recording the time and correctness of the buttons clicked.

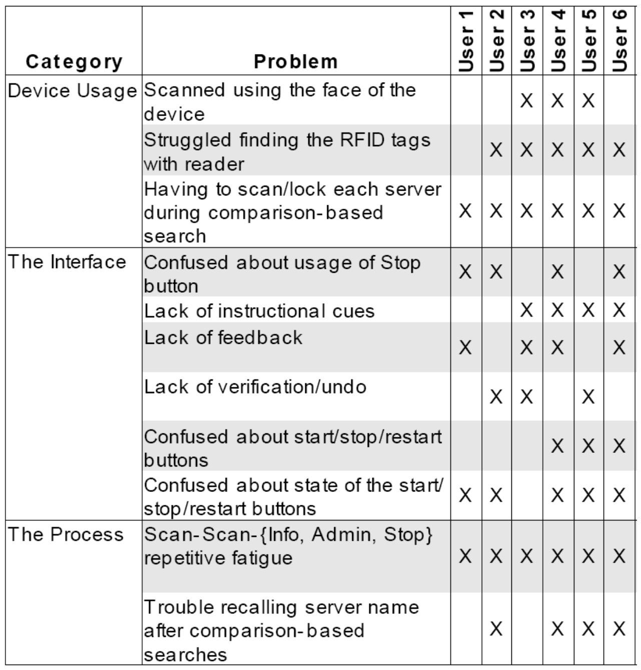
There were some small differences between the prototype and the actual device. Since the resolution was fixed in the device description, and the resolution of the touchscreen was more compact, the device was approximately half the physical size that was intended. The group chose to use a touch-pen to overcome the difference in size and protect the touchscreen from damage.

The graduate student for their group collected his experimental data, and created a plot of time against missed clicks (Fig. 3.3) to determine the probable cause of frustration or misclicks. Each task was identified by number, while letters represent notes from observers about actions the user took, questions the user asked, or visual cues of the user's emotion. This allowed the graduate student to discern the problems with the device that caused the most errors.

### 3.3.2 Location Aware Remote Control (LARC)

Another project with a tactile aspect to their prototype was the Location Aware Remote Control (LARC). LARC is a server maintenance application which used Radio Frequency Identification (RFID) cards to decide which server it was controlling. The application used large icons and very few buttons on the touchscreen to avoid errors. The most interesting aspect was the use of real RFID cards which represented the user selecting different servers to make modifications.

The LARC developers chose to implement a "Wizard of Oz" prototype. The user used an identification card to log in, then moved the RFID reader across a fake server rack with



| Category | Problem | User 1 | User 2 | User 3 | User 4 | User 5 | User 6 |
|---|---|---|---|---|---|---|---|
| Device Usage | Scanned using the face of the device | | | X | X | X | |
| | Struggled finding the RFID tags with reader | | X | X | X | X | X |
| | Having to scan/lock each server during comparison-based search | X | X | X | X | X | X |
| The Interface | Confused about usage of Stop button | X | X | | X | | X |
| | Lack of instructional cues | | | X | X | X | X |
| | Lack of feedback | X | | X | X | | X |
| | Lack of verification/undo | | X | X | | X | |
| | Confused about start/stop/restart buttons | | | | X | X | X |
| | Confused about state of the start/stop/restart buttons | X | X | | X | X | X |
| The Process | Scan-Scan-{Info, Admin, Stop} repetitive fatigue | X | X | X | X | X | X |
| | Trouble recalling server name after comparison-based searches | | X | | X | X | X |

**Figure 3.4:** User analysis for LARC.

transponders. Then, each user performed a series of tasks, watching the computer screen for responses, and telling one of the moderators which buttons he pushed. The moderator performed the actions on the computer, which allowed the user to see the response. This method of prototyping was chosen to allow the user to understand how the device feels when operated in its intended environment. Many test users were initially skeptical about moving the RFID reader around instead of working directly with the laptop, but after they saw the screen change based on which system they were examining, they recognized the importance of the physical aspect of this device.

The success of this user testing was primarily due to the graduate student's preparation. Not only did they create a background program to record automated timings of events (much like Mini-Mote in Fig 3.3), the graduate student also was careful to examine the device before the experiment, and make predictions about potential user errors. The graduate student had prepared observational cues for the group members to look for. This guided their understanding to find the real problems users faced (Fig. 3.4).

After seeing the results, especially about repeating the "lock" step multiple times, the group members recognized that they could improve their device. One of the comments made at the end of the course was the desire to improve their design and test again. A similar sentiment was shared by other groups as well, including the MP3 Stingray project.

### 3.3.3 MP3 Stingray

A third group decided to create an interface for an MP3 player, called the MP3 Stingray. As a common choice for design projects, groups that choose MP3 players are asked to attempt something novel with their device. To meet this requirement, they created a randomized playlist generator which selected from sublists based on genre, artist, album, and existing customized playlists. The generator allows the user to create a mix timeline where the device chooses randomly from the sublists associated for that time.

Although their task seemed to be very straightforward, the small size of the TDA and the inexperience of the group members meant that they were required to get past a number of usability issues. During the Heuristic Evaluations, they were asked to make adjustments based on almost every heuristic, and though they made significant improvements between

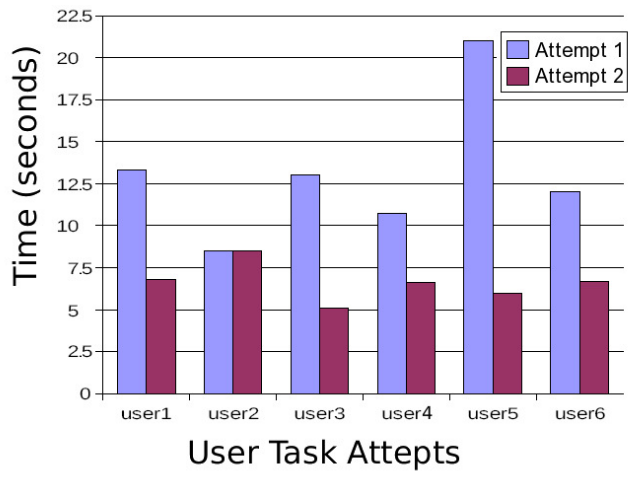


**Figure 3.5:** Task completion time for MP3 Stingray.

their low-fidelity prototype and their high-fidelity prototype, the User Testing revealed additional problems with their device.

During their observations, they noticed that each user took significantly longer to complete the tasks than expected. Although users were quicker the second time (Fig. 3.5), evaluations based on the graduate student and group members' observations indicated that certain tasks were not intuitive. The users had simply memorized the steps to complete those tasks, instead of basing their actions on the ambiguous visual cues from the device.

Despite all of MP3 Stingray's challenges, the group members seemed more enlightened at the end of the course; during the exit interview, all of the group members were very intense in their praise of the graduate students' feedback and the perspective gained from user testing. We believe that experiences like this will inspire students to practice usability evaluation during the design process in the future.

### 3.3.4 Other projects

In the original article[13], we focused on three of the seven projects. These three projects provided some of the best specific examples of student's learning from the testing. A short description of the other four projects are presented here, and the interviews are presented in Appendix B.

#### 3.3.4.1 Portable Beer Pong Scorer

The Portable Beer Pong Scorer was intended for easy use while intoxicated. It allowed users to keep score of the common drinking game Beer Pong.

One difficulty faced was setting up the user testing. The students who were asked to participate in user testing were from introductory courses, so many were under the legal drinking age. In addition, University policy prohibited alcohol use on campus. Therefore, the unit tests done for course credit used sober subjects.

Like Mini-mote, Portable Beer Pong Scorer was put onto a real device. In their case, they used a smart phone, which inspired the use of Android for later iterations of the course (see Chapter 4).

#### 3.3.4.2 Groove on the Move

Like MP3 Stingray, Groove on the Move created an MP3 player designed for a user who is moving around. To provide a distinct aspect for the device, the group created some gesture-based controls for the device. Users can select songs to be queued or songs to be added to a "party mode."

This group gained some benefit from the evaluation; however, they were focused on the development of their device rather than the interaction with the graduate students. They said that waiting for the graduate student feedback , especially from the Heuristic Evaluation, slowed their progress down.

#### 3.3.4.3 Run Tracker

The RunTracker application was intended to track statistics and distance for a runner wearing the device. The primary user for the device would be a person exercising outdoors.

During the final interview, this group showed enthusiasm for their project and the interaction with the graduate students. The group's first interaction with a graduate student led them to focus on the user's experience for their design. Ultimately, their design led to an experiment that provided them positive usability feedback. One of their responses in the interview was that they wished they had time to complete the project.

#### 3.3.4.4 TDA Maps

The TDA Map group created a GPS device which would provide users with location, direction and speed, along with nearby points of interest. During the interviews, we specified that according to the assignment description, they must provide a unique aspect to their device. Ultimately, their device was similar to modern vehicle GPS systems, but had a sub-par design.

This group was also the most negative about the experience. Many times during the interview they complained about waiting on feedback from the graduate students delaying their production. After the interview, the interviewers discussed whether there was any indication that these students understood the intent of the course.

## 3.4 Evaluation and Conclusions

Our goal was to provide two distinct experiences through interacting roles by members of the combined courses. The undergraduate students took on the role of User Interface design and implementation, while the graduate students provided the role of evaluation, usability testing and analysis. A combined graduate and undergraduate HCI course benefited both courses and the student roles from those courses. Each graduate student participated in the development of several UIs and had the opportunity to conduct usability testing on a unique UI. Each undergraduate group received feedback on their UI design from several graduate students, then helped with the usability testing of their UI. Conducting the combined course in such a manner allowed the undergraduate and graduate students to interact with each other, exposing the undergraduate students to graduate school life. We hope that this exposure may encourage more undergraduate students to continue their education on to graduate school.

Independent exit interviews were conducted with each graduate student and with each undergraduate group. The questions and notes from these interviews are in Appendix B and Appendix C Almost every group talked enthusiastically about the usability testing, the graduate students' evaluations and what they learned from those. When asked about the graduates, one group provided a very concise description:

> "They [the graduates] had good ideas and helped make our project a little better. They were sort of like the customer: they had a unique perspective that we maybe wouldn't have come up with ourselves."

This quotation reflects the overall theme of the undergraduate group responses: All but one of the undergraduate student groups were able to provide a specific positive impact in either their design process or focus. When asked about limitations, four of the groups indicated no limitations from the graduate interaction, and three groups indicated “minor slowdowns” and “delays while waiting for [graduate students].”

Many of the graduate students’ user testing observations were overshadowed by the insight that the undergraduate students gained from simply participating in usability testing. The group members did not realize how important simple usability issues were before users began to get confused. The prototyping aspect of the course also gave the students specific goals. Instead of being finished with the projects, all of the groups had some aspect to their project which was “faked” to respond to the user’s commands, but represented what the device would eventually do. A few of the undergraduate students mentioned their desire to complete the software.

The graduate student experiences helped them better understand the evaluation process, since they were practicing on developing projects, instead of completed devices. They also experienced many of the pitfalls that a released product could not provide, including: delays, incomplete prototypes, and uncooperative developers. Despite these obstacles, all seven of the graduates produced detailed, interesting documentation regarding the analysis of their group projects. Even the two graduate students who had taken the undergraduate course in previous semesters commented that they benefited from taking the graduate course.

The other aspect of the course that the group members discussed was the graduates’ research presentations. Many undergraduates found the research presentations interesting, and commented on them during the exit interviews. One student said that he “didn’t even know that there was such a thing as Tangible User Interfaces” until he heard the graduate’s presentation.

The evidence from interviews indicate that these courses imparted deeper understanding to the students. The response was well above our expectations. Therefore, we were successful in meeting our goal: two truly distinct, yet combined courses. We believe that this is largely due to the experiences of the interactions between the groups and the graduate students, and the usability testing on high-fidelity prototypes. These two features, student interaction and high-fidelity user testing, were possible exclusively through the use of combined courses following an experiential learning model.

## 3.5 Discussion

Since this course closely followed Kolb’s experiential learning model, we believe it provides potential answers to our research questions. First, the course provided a unique user testing experience for both testers and developers. Second, the instructor gained insight into the amount of additional work that is involved in managing two interactive experience-based courses. Finally, the three-phase course provided a gradual means for students to update their mental models about user interface design, while mitigating negative attitudes. The growth of the course into the course presented in Chapter 4 is a testament to the success of the experiences created through this course.

# Chapter 4

# Adding Unit Testing to an HCI Course

The material for this chapter was adapted from *Adding Unit Test Experience to a Usability Centered Project Course* [14]. As described, the course was an extension to the course presented in Chapter 3. The addition of Unit Testing to the HCI course, as described here, was attempted in the Spring semester of 2013. Since much of the introductory material of the original paper was already presented in Chapter 1, it has been removed from this chapter.

## 4.1 Introduction

This unit testing research is a step in the continual development of the combined graduate and undergraduate, multi-role Human Computer Interaction courses [63]. As the course grew, it became necessary for the projects to be more complete. To complement the usability testing in the course, we decided to add an emphasis on functionality through unit testing. We used Kolb's Experiential Learning Model (ELM) [42] to add functional testing to the course in order to improve the quality of the projects and provide a more complete testing experience.

The courses began as a combined graduate and undergraduate effort intended to provide a professional experience for both the graduate students as usability experts and the undergraduate students as developers [13]. As the original graduate and undergraduate courses grew, additional elements were added. The original courses' final product was a fictional hardware product with intentional restraints. With the advent of Google's Android Software Development Kit (Android SDK) [28], students in the course began to develop for a portable device. As part of a collaborative Citizen Science project, real customers were added to the course. These customers were scientists with a known desire pushing students to design to a specific need rather than their own fictional project [54].

It was the Citizen Science project that drove the need for software quality improvement.

In the original iterations of the undergraduate course, the focus was on the usability of the project, and an incomplete project was still an appropriate learning scenario. With real customers, students had an obligation to have functioning code, and after the initial effort, it was decided that more emphasis on that functionality must be given. To accomplish this goal, unit testing was added as a new element to the course.

### 4.1.1 Unit Testing in Education

One of the common testing methodologies is unit testing. Unit testing is a testing approach with the following properties: Simplicity, Independence, and Documentation[6]. Tests focus on a unit, or single piece, of the overall functionality within the program, ensuring that they work independently, for smoother integration. The unit can vary widely in size, from a single function, collective behavior of methods within a class, or a group of classes which work together. By being simple and independent, the result of a unit test should be consistent for repeatability. Unit testing is often automated, and a benefit of creating automated tests is that the purposeful thought of creating the tests causes students to avoid and discover errors quickly. JUnit is a tool for unit test development [7], and is supported by the Android SDK [28]; therefore, it is an appropriate choice for testing the Android projects in this course.

Introducing unit testing in an academic environment often involves introducing tools to improve student experience. The JUG tool allows students to see unit test results to provide feedback in their own code [15]. Saff introduced a tool to allow their IDE to perform testing continuously, much like current IDEs check for compilation errors while the program is being written [67]. These tools encourage experiences with unit testing, and demonstrate the benefit of testing to development. However, these two tools only assist with existing unit tests, they do not provide students assistance in writing unit tests.

When introducing the construction of unit tests, students need more guidance and encouragement. Kaner and Padmanabhan explored experiential learning of unit tests, emphasizing instructor enthusiasm and reporting on some areas where students become confused [38]. They identified the major confusion elements were "applying a standard procedure", "Figuring out (and explaining the choice of) boundary values" and "Identifying risks and associated error-revealing classes." Their solution to the confusion was to increase the models and example of testing.

Some other researchers introduced peer testing to provide enthusiasm for testing. Smith's introductory course used competitive peer-testing to encourage testing practices [69]. Students would get points for "breaking" their peers' code by writing unit tests for appropriately difficult cases. Smith's work was based on Clark, who introduced peer group testing in a project course [17]. Groups were assigned a few students from other groups to write their tests. Although this is a useful way of introducing unit testing to projects, we felt that students learning about another group's distinct project would introduce a much larger time requirement than we intended. We also wanted students to consider the difference between the functional and interface aspects of their project.

### 4.1.2 Research Questions

When introducing unit testing to the course, our primary research question was:

**Can Unit Testing improve the quality of Human Computer Interaction projects?**

However, after we discovered that students were not participating in the unit testing aspects of the course, we decided to expand the research to two other questions.

**When introducing unit testing, what additional steps must be taken to ensure a positive learning experience?**

**What potential for regression of students' unit testing model is possible, and how can that potential be mitigated?**

We decided to refocus our research on learning experiences and model regression because informal discussions during the course revealed prejudice against unit testing and its practicality. Our goal is to minimize the possibility of students regressing to those mindsets.

## 4.2 Method

The methodology for this research was similar to the methodology used for earlier additions to the course [13]. Students were given assignments that encouraged or required them to actively experience unit testing. The graduate students collected project data on effectiveness, and the undergraduate students participated in a survey and interview at the end of the course.

### 4.2.1 Initial Learning Activity

The ELM indicates that Reflective Observation occurs after feedback. To accomplish the first cycle, we required students to participate in an initial unit test activity. To help students get started Android programming, the course required building sample Android projects. The Android Software Development Kit also contains support for JUnit testing, including tutorials.

The first unit testing activity for students was to complete the Android JUnit test tutorials [28], followed by another sample project that extended the ideas of those tutorials. These sample projects were expected to take a few hours and ensure that students had their systems set up to handle processing unit tests. These assignments were given to each individual, not to the group, so that each person would be responsible for writing tests.

### 4.2.2 Unit Testing Lecture

The next step in the course was a lecture discussing unit testing and Test Driven Development (TDD) (for a more detailed description of these topics, see Section 2.2.2. This lecture encouraged abstract conceptualization (AC) about the role unit testing plays in software development. The lecture built off the activities the students participated in, and offered some indirect feedback to the tutorials.

First, unit testing and TDD are defined and provided as an analogy. Second, a demonstration provided an example of some pitfalls to avoid when using Android's JUnit testing, building off the examples the students did. Next, the lecture reminded students of an indirect experience with unit testing most of them had earlier in the curriculum [15]. Finally, the lecture emphasized the role of the developers as testers and verifiers.

### 4.2.3 Assigned Testing

At the next stage of the course, students had a midterm interview with the instructor to narrow their focus to obtainable goals. After their initial experience into unit testing, but before they had begun major development into their project, students were assigned to build automated unit tests for one activity (screen) in their application. They were also asked to leave a different activity without automated testing as a point of comparison. Unit tests for other areas were left to their discretion.

The goal of dividing the project into tested and untested aspects was to demonstrate the benefit of unit tests. The activities in the programs had to be divided based on the individual project, but we took steps to find elements of similar difficulty. We did not require students to employ TDD since the techniques are usually developed after programmers are comfortable with test writing. We did encourage experienced testers to use TDD and asked all students questions about their experiences with TDD.

### 4.2.4 Usability Testing

Our previous work suggests that students have a certain satisfaction culminating the project with usability testing. From that idea, we decided to treat it as a functional release. The usability testing portion of the course provided a deadline for the project, which is the culmination of the project work. As with many software projects, during final testing, groups have technical problems with code working incorrectly or missing components. In previous semesters, the groups would fix the bugs for later usability tests and the graduate students may make a note distinguishing the "pre-fix" test subjects from the "post-fix" subjects.

To identify and document these functional problems, an additional assignment was added to the graduate students. They were to develop appropriate bug report forms for the different projects, and collect data concerning technical problems and functionality issues during usability testing. Both the graduate students and undergraduate students were responsible for filling out the bug reports. The completed bug reports can be found in Appendix F.

### 4.2.5 Learning Evaluation

We used two methods of data collection to evaluate the learning objectives of the unit testing addition to the course. First, an interview was done with each group to get their perspective of the unit testing aspects of the course. The interview allowed students to provide free-form feedback and included discussion and questions related to their experiences. Students were quite frank in the discussion, and we discovered that some groups did not do the assigned unit testing.

Students also filled out an anonymous survey, including a self-reported inventory of their experience. The survey also included some knowledge verification questions, used to determine what aspects of unit testing students took away from the experience. Finally, the survey asked questions about their expected future use of unit testing, to ground their opinions of unit testing. This allowed us to compare groups and evaluate their answers to questions about usability testing with respect to their experience before and during the course.

## 4.3 Results

The results for this research showed that the motivation for experiencing unit tests was insufficient for most of the groups. Unfortunately, that left very little data to evaluate the effectiveness of the experiential learning. The surveys and interview questions did provide some insight into the student's mental model, and we hope that this can be used for a more successful second attempt.

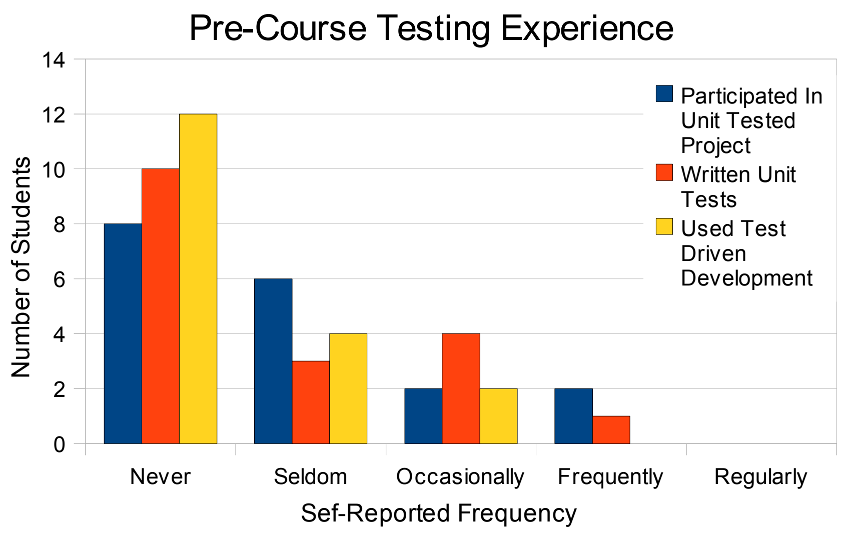


**Figure 4.1:** Prior Unit Testing Experience

### 4.3.1 Interviews

Like the group interviews from Chapter 3, we received very frank feedback from the individual interviews. The interview questions and transcribed answers can be found in Appendix D.

One of the groups, Stream Features, performed a significant amount of automated unit testing work. Two other groups also included unit tests, though their implementations were minimal. The final two groups did not include unit testing. One indicated that they could not unit test because they had too many abstract objects in their code. The other group did not create unit tests or participate in unit testing, since their project was incomplete.

The Stream Features project included strong database interaction, and they used unit tests to verify that their database connections worked properly. They employed TDD, writing their database tests and functions before they were used in the rest of the project. This was also the only group whose bug reports were purely cosmetic. The interview revealed that most of the group members were comfortable with unit testing from a previous course. They expressed the perspective "... sometimes JUnit testing [feels like] a lot of work that doesn't pay off." That perspective given by a successful group shed some light on the other groups' challenges with unit testing.

One member of the Lichen group created a single unit test, and later discussion indicated

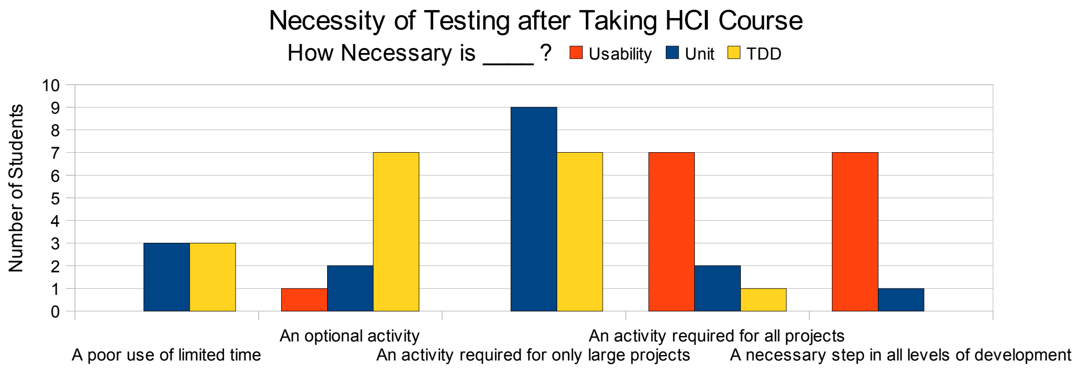


**Figure 4.2:** Views on the Necessity of Testing

that they had difficulty creating tests for more complex actions. The student who wrote the single unit test had positive feedback about the process and indicated regret at not writing the tests sooner. They also indicated that the unit test ran much faster than loading the emulator. These students did not have the same course experience as the Stream Features group and expressed a desire for more experience.

The Tracker Team wrote their unit tests after the usability tests were complete but before the interview. The primary reason they wrote the unit tests was to avoid a grade penalty.

Although the interviews seemed to indicate a positive experience with those students who used unit testing, we believe there is potential for stronger instructor interaction. The students who did not use unit testing struggled with time and had many more problems in their implementations.

### 4.3.2 Surveys

After the course and interviews were complete, students were requested to take an anonymous survey about their work in the course. Of the 21 students enrolled in the course, 18 responded to the survey. The survey included questions about their own history with unit testing, their views on testing, and the extent of their activity within their group. The survey included open-ended questions designed to expand on the multiple choice answers from other sections. There were also five questions which attempted to evaluate student understanding of the distinction between the purposes of unit tests and usability tests. See Appendix E for the questions from the survey.

#### 4.3.2.1 History With Unit Testing

In terms of the ELM, most of the students have an immature mental model of unit testing. As shown in Figure 4.1, only 1 student reported writing unit tests frequently, and more than half the students who responded had never written unit tests for a project. Strangely, one student claimed to have used TDD, but indicated they had not participated in a project with unit testing or written unit tests. All students were required to take a Team Software Project course, however, as indicated by the interviews, not all iterations of that course introduced unit testing.

In Figure 4.2, we see that after the HCI course, students have a near unanimous positive view of the necessity of usability testing. In contrast, the common view of unit testing is that it is only useful in large projects. The phrasing “an activity required for only large projects” was used as a common answer when discussing automated testing casually with students. It was also a common perception among our students, as half the students responded with that answer. Students have a view of TDD that indicates it is less necessary than unit testing, with most students indicating that it is optional or a poor use of time.

The survey asked 6 questions about students anticipated use, indicating whether students would use or recommend different testing methodologies in the future. Overall, students support usability testing and favor unit testing in a professional environment. In Figure 4.3, this is represented by the Green and Light Blue bars. However, for personal use (the Dark Blue and Yellow bars), students are more neutral. Overall, students were neutral toward TDD in both personal and professional used (Light Blue and Maroon).

#### 4.3.2.2 Inter-Group Activity

In most of the teams except Lichen, one of the team members claimed responsibility for writing tests. In the interview, only one of the members of the Lichen group reported writing unit tests, and based on the activity questions, we believe that student was one of the 3 that did not answer the survey. As discussed earlier, only three teams wrote unit tests, and of those, only one used TDD. One of the teams wrote their tests after usability testing, which means that they did not get the intended finalizing feedback of the usability tests as they pertained to the unit tests.

Responses to functionality problem questions indicated that bugs were discovered in program elements without unit tests. To avoid skew; however, we should only consider the two groups that wrote unit tests before usability testing. In the Stream Features group, the group that employed TDD, all four students responded that no functionality problems were found in unit tested code. Of the three students that responded in the Lichen group, two indicated that no problems were found in unit tested code, while the third indicated "a few" problems found in unit tested code, while "most" problems were found in untested code.

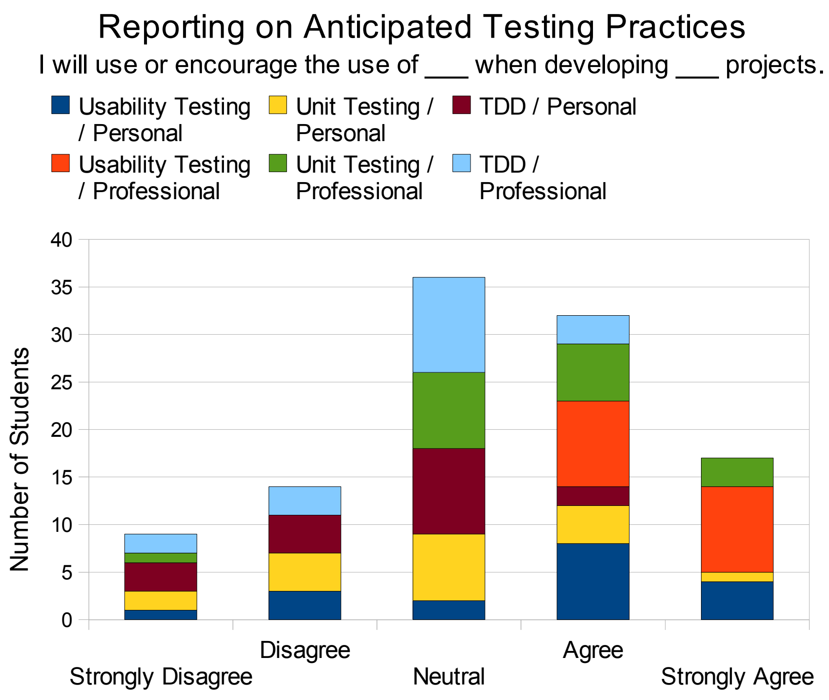


**Figure 4.3:** Responses to Future Testing Scenarios

#### 4.3.2.3 Free-Form Responses

The free-form questions on the survey provided insight into the students roles and opinions of unit testing.

In the Stream Features group, one of the students indicated that they did not write a significant amount of code for any program element, and when asked about which elements were unit tested, they wrote “I have no idea.” Within the Stream Features group, that student gave the only negative response to unit testing in the open-ended questions. The other three students wrote about positive experiences using unit testing to improve their code. In fact, one of the students from Stream Features indicated another element that they thought should have been unit tested.

In the Beach group (which did not include unit tests), students’ free-form survey responses indicated that their choice to use anonymous classes prevented unit testing, but that they each had a desire to include unit testing. This group also had the only student who reported “frequently” using TDD before the course. This student also provided the only strongly positive experience of unit testing outside the Stream Features group.

The last question on the survey asked for additional comments, which provided a large number of negative comments toward unit testing. One student was concerned with writing tests for GUI elements when they did not have those elements complete until the last minute. A similar statement indicated that unit testing and TDD “have their place”, but are “difficult to use. . . in projects that are largely GUI based.” Another student said that unit testing “seemed like a very slow process and difficult to do”, though their indications earlier in the survey indicated that they had not done unit testing before or during the course. A third student commented that they did not understand why unit testing was needed, because it only found a “couple minor bugs”, and “we can do most of our debugging tests by hand.” As described in the analysis, these comments unknowingly dismiss some major benefits of unit testing.

#### 4.3.2.4 Understanding Unit vs. Usability Testing

The questions which evaluated student understanding of the purpose for unit tests and usability tests were scenarios of a business website. The questions were ordered by increasing complexity. The premise was that students need to verify that they had completed the described work with appropriate testing. They were to select the type(s) of testing that would be most appropriate.

Most students answered the first two questions and the final question correctly, though some students answered “Both unit and usability tests” when one of the types of testing was inappropriate or excessive.

The third question was difficult for students, it concerned Autofill of street names. The idea behind this question is that students must identify functional aspect of determining a street name from a partial string, while also recognizing that the way an autofill presents information is a usability issue. Only 6 of the 18 students identified both the functional and usability aspects of this scenario.

The fourth question concerned verifying that the text of the website matched a brochure. In this case, unit testing is inappropriate because verifying the text within the unit test would be just as cumbersome as manually checking the site. Only 6 students identified the redundancy of a unit test in this scenario.

Although many of the students answered the easier questions correctly, the students who did not answer those correctly were also the students who offered the most negative comments in the open ended section of the survey.

## 4.4 Analysis

We expected that adding a unit testing experience to a project based HCI course would flow naturally. Usability testing was already a focus of the course, and functionality had been a problem in the past. We speculated that adding functionality checks would improve the overall product. Instead, we discovered that students expected unit testing to be a negative experience and resisted.

The most difficult challenge for the course was ensuring that students experienced unit testing. Students resisted each stage of the process. During the lecture, an informal show of hands indicated that less than a third of the students had done the initial tutorial activity. After the lecture, a few students began arguing about the practicality of unit tests, eventually acknowledging that they had never participated in a project with unit testing. Near the end of the semester, during usability testing, only 2 of the 5 projects had used unit testing.

The successes of the groups that participated in unit testing and the opinions provided by all the students helped formulate answers to our research questions:

**Can Unit Testing improve the quality of Human Computer Interaction projects?**

Despite the limited data in regards to the effectiveness of unit testing, the group that did participate fully in the unit testing showed remarkable results, with no functional problems in their program. Their project also showed meaningful results in the usability tests, since they could function without problems, they were able to discover more subtle usability issues than could have been possible with a partially functioning program. In addition, their program was available for the Citizen Science project that it was intended, providing a positive experience for the students participating.

**When introducing unit testing, what additional steps must be taken to ensure a positive learning experience?**

A potential problem with the structure of the course may have been caused by adding additional experience to a high-intensity project course. Students treated the unit testing portion of the course as if it were a minimal part of the project, while completing the project was their primary focus. This perceptual divide implying that unit testing would delay project completion may be a result of de-emphasis at the curricular level. It may be the case that a curricular unit testing focus, as proposed by Goldwasser, could improve student attitudes [27]. In addition, many student's mental model of unit testing causes them to believe that test development would delay interface development.

One of the primary means of ensuring a positive learning experience is to emphasize the more concrete benefits of unit testing. For instance, the Lichen group discovered that unit tests would run more quickly than the emulator. It is also important to include warnings and suggestions, such as those provided by the Beach app to avoid anonymous classes. Finally, we must encourage the idea that TDD is not only possible, but straightforward for a GUI based application, so long as there is low coupling between the data and the GUI element (something the Android SDK strongly supports [28]). Emphasis on these three elements of testing Android programs could prepare and encourage students to have a positive experience.

**What potential for regression of students' unit testing model is possible, and how can that potential be mitigated?**

Based on the differences between responses of groups that experienced unit testing and those that did not, the biggest potential for model regression is students who choose not to fully participate in the experience. To mitigate this, the instructor could place a firmer emphasis on the requirement of unit testing as part of the course and request that students write tests for code other group members have produced. It is often easier to convince a programmer to test someone else's code rather than try to convince them that their code requires testing [17].

The other possible regression could be caused by confusion of the purpose of unit testing [38]. If a student's conviction is that unit testing is only being done for trivial or subjective aspects of a program, it can cause them to regress. The survey responses indicated that students had a belief that the only purpose of the unit test was to discover "minor bugs" and that "hand testing" works better. An emphasis on the scalability and practicality of automated testing (for instance, that it is faster than loading the emulator) could encourage more participation. In addition, students might relate to an analogy of unit test creation as a series of thought exercises about the code. Another area of confusion might be the tutorial example, its simplicity implies that unit tests must be done for trivial aspects of a program. To mitigate this, the instructor should demonstrate that the tutorial is a simplistic example, and follow with a more complex, yet concrete example.

## 4.5 Conclusions

Unit testing can improve the quality of projects in an HCI course, but the challenge lies in convincing the students that unit testing is valuable. We found students developed a positive attitude after experiencing usability testing. A stronger curricular and instructor emphasis on the benefits of unit testing could provide a similar attitude change. Within our course, the next step is to put structure and emphasis around the unit testing element of the course. Ultimately, the struggle of including unit testing into the programmer's knowledge base is that of participating fully in the experience.

# Chapter 5

# Conclusions

## 5.1 Review of Questions

Each of the three research opportunities provided insight into the way Experiential Learning can be applied within Computer Science. By relating the experiences and expectations to Kolb's four stages of learning, we can evaluate the successes and opportunities for development. It is our hope that the pedagogical research done here can enhance the experiences provided in a Computer Science curriculum.

The first aspect, Concrete Experience (CE), is present at every level, from first year students through graduate students, driving the classroom learning through activity. Through provided testing and feedback, undergraduate students are encouraged to participate in

Reflective Observation (RO) stage of learning. The higher level courses are then able to focus on Abstract Conceptualization (AC) by generalizing key experiences into heuristics and practices. Finally, the cyclic nature of the presented courses offer opportunity for students to Actively Experiment (AE) with their new-found knowledge, reinforcing their updated models.

**How can the Experiential Learning Model (ELM) be used to incorporate testing experiences to computer programming courses?**

In Chapter 2, the experiences in the course focused mainly on correct programming implementation through instructor developed unit testing. Students were motivated by their grade to participate in the Concrete Experience (CE) of the course, but needed fast feedback to allow Reflective Observation (RO). The JUG tool allowed the instructor to encourage regular reflection, combining that feedback with lecture material for Abstract Conceptualization (AC), allowing students to gradually build their mental model. In addition, since JUG is rooted in JUnit, students had an additional Concrete Experience (CE) through exposure to industrial testing techniques.

In Chapter 3, we discovered that a multi-role course can help graduate and undergraduate students learn high level concepts of Human-Computer Interaction. The focus of the course was on Concrete Experience (CE) and Abstract Conceptualization (AC), which were successfully implemented. Interacting with various roles allowed students to experience a professional project development environment. Undergraduates then analyzed the experience of a Heuristic Evaluation alongside a lecture on common usability issues, to improve their design for Usability Testing. Finally, the undergraduate students could get the benefit of a real usability test, while graduate students gained experience testing and analyzing a product.

Chapter 4 allows us to see the potential in Unit Testing experiences as part of a project course. Students were expected to participate in two Unit Testing experiences (CE), tying into a lecture for Abstract Conceptualization (AC). Unfortunately, few of the students participated in both exercises, and our plans did not expect high level undergraduate students to be as resistant to implementing Unit Testing in the course. Although there were problems in the course, the students that did participate in unit testing showed a distinct view of Unit Testing from those who did not. We can use the differences between student perceptions of Unit Testing to help answer our other two research questions.

**When introducing testing experiences, what additional steps must be taken to ensure a positive learning experience?**

In the lower level courses, the most important step that the instructor can take is to provide strong feedback quickly. Students who have a desire to grow in programming respond well to fast feedback, and recognize the importance of reinforcing concepts. The largest obstacles that an instructor must overcome are incorrect and incomplete tests.

Incorrect tests cause students to distrust requirements, and incomplete tests give students the impression that they do not have to complete the program. Finally, the instructor must be able to balance automated grading with supplemental grading criteria (i.e. coding standards, time complexity analysis, algorithm correctness). With the JUG system, we provided feedback twice for each assignment, one with only automated results, and the second with subjective grade feedback.

In the original HCI class, students were very receptive to Usability Evaluations and Testing. The instructor for a mixed course was required to spend more time preparing to ensure the interactions between graduate students and undergraduate students would be positive for the learning model. In addition, the instructor had to make managerial decisions about the effectiveness of graduate students' proposed testing methods in response to student projects. Later in the course, as additional elements were added, the preparation and planning also increased. The addition of Unit Testing was designed to be a simple way to improve the quality of the final products, however, we discovered that additional steps are needed to increase student enthusiasm for Unit Testing practices.

**What potential for model regression is possible, and how can that potential be mitigated?**

In the introductory courses, we found that tight cycles of experience and feedback create potential for model regression through discouragement. A student who misses a program, or falls behind in the lecture material has difficulty incorporating their incomplete experiences into their model. In the Data Structures course, where programs build strongly on previous knowledge, it is important that students be given the opportunity to catch up. The method we used to mitigate this problem was the "resubmit"—allowing students to get a small penalty for a late assignment yet keep the momentum of the course and participate in the experience based learning.

In the Usability Evaluation and Testing for the HCI class, the potential for regression is when students dismiss the design evaluations. One excuse that students can use is that the graduate students doing heuristic evaluations are amateur "experts". The way to mitigate this excuse is to help them recognize that even without experience the graduate students cite recognized usability principles that their interface is not following. The other potential regression is to deny Usability Testing results based on the usability test subjects. Often students see the design flaws but the solution to a usability problem is not obvious, so developers resort to a "can only work one way" mentality regressing their mental model about that particular usability issue. Mitigating this regression is difficult, but providing small usability improvements or alternatives and reinforcing the importance of a positive user experience can help.

The Unit Testing experience in the HCI class had the most regression, primarily because students chose not to participate. When the student intentionally ignores the minimal experience they participate in, they have nothing to reflect upon or conceptualize. The other possible regression can be caused by confusion of the purpose of unit testing. If a student is convinced that unit testing is only effective if done for trivial or subjective aspects of a program, it can cause them to regress. To mitigate this, we speculate that an instructor could introduce various improvements to the course:

- † Place a firmer emphasis on the requirement of unit testing as part of the course.
- † Request that students write tests for code that other students are writing (similar to Clark's method [17])
- † Provide analogy of early tutorials to "Hello World" programs, to avoid students mentally dismissing unit testing as "simplistic"
- † Provide additional examples of unit testing with increased complexity and concrete elements

These mitigation methods can provide the foundation to encourage the developers to participate in the experience, ultimately improving their model of unit testing.

## 5.2 Goal-Driven Test Experiences

Overall, the results of this research shows the benefits of using the experiential learning model in the computer science classroom. Each of the course studies demonstrate that students can gain practical knowledge through experience involving testing. The most important aspect of these experiences are that they are each driven by instructor goals. Along with their individual goals, they have a shared goal of exposure and enthusiasm for the subject material. The major goals for the three research projects are based on testing experiences.

In the introductory programming courses, the goal was to provide more practice. That goal drove us to create an automated testing tool that made generating assignment tests easier. Finally, when introducing Unit Testing to an HCI project, the goal was to improve project software quality. Those students that participated fully in the unit testing experience had no functionality errors.

The goal from the Usability Testing was to emphasize the practice and importance of recognizing the user experience. This goal lead us to introduce various testing experiences for both graduate students and undergraduate students. Students evaluating each other can practice what they are learning while demonstrating their techniques to others.

The course goals help define the specific experience, however, the overall goal is to inspire the students. Unit Testing low level code inspired students to consider all the functionality when programming. Usability Testing inspired students to think about usability in their software. Writing Unit Tests inspires students to desire high quality in their completed software.

## 5.3 Conclusions

In the presented research, providing testing was driven by course-level goals. Our research questions focused on using Kolb's Experiential Learning Model (ELM) to meet those goals. Overall, the three courses offered valuable information for experienced-based learning practices.

Although the ELM can be used to good effect, there are difficulties associated with using it to promote testing in a computer science classroom. Integrating test experiences required more effort on the part of the instructor; however, there were ways, especially with automation, to reduce that effort. It is also important to design a course with the understanding of the appropriate goal of test experiences. Ultimately, the most important part of introducing testing to later courses is mitigating model regression. When using the ELM, the CS instructor must provide the opportunity, emphasize the importance, keep students involved, and offer direction in the face of failures.

The Experiential Learning Model provides a means of incorporating new computer science material into a classroom. Throughout these course-based research projects, we used experiential learning to include testing practices into various courses in a computer science curriculum. We were able to improve feedback to students and provide testing experiences similar to industry. Our results showed that the course changes improved student understanding. When examined together, these projects provided insight into building strong testing practices into a curriculum. We used the ELM to provide testing experiences; however, understanding experiential learning can provide insights into many other aspects of the Computer Science curriculum.

# References

[1] A. M. Aiken. MOSS: A system for detecting software plagiarism. `http://theory.stanford.edu/~aiken/moss/`, 1994.

[2] A. Allowatt and S. H. Edwards. IDE support for test-driven development and automated grading in both Java and C++. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology Exchange*, Eclipse '05, pages 100–104, New York, NY, USA, 2005. ACM.

[3] R. Arora, P. Bangalore, and M. Mernik. Developing scientific applications using generative programming. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, SECSE '09, pages 51–58, Washington, DC, USA, 2009. IEEE Computer Society.

[4] G. Attardi, A. Cisternino, and A. Kennedy. CodeBricks: Code fragments as building blocks. In *Proceedings of the 2003 ACM SIGPLAN Workshop on Partial Evaluation*

*and Semantics-Based Program Manipulation*, PEPM '03, pages 66–74, New York, NY, USA, 2003. ACM.

[5] E. G. Barriocanal, M. Ángel Sicilia Urbán, I. A. Cuevas, and P. D. Pérez. An experience in integrating automated unit testing practices in an introductory programming course. *SIGCSE Bull.*, 34(4):125–128, dec 2002.

[6] Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[7] K. Beck. JUnit: Resources for test driven development. `http://www.junit.org/`, 1994.

[8] A. Begel and B. Simon. Struggles of new college graduates in their first software development job. In *SIGCSE '08: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, pages 226–230, New York, NY, USA, 2008. ACM.

[9] B. Beizer. *Software System Testing and Quality Assurance*. Van Nostrand Reinhold Co., New York, NY, USA, 1984.

[10] A. Bertolino. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, FOSE '07, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society.

[11] S. Brandle. Automated grading of student programming assignments. *J. Comput. Small Coll.*, 25:83–84, Oct 2009.

[12] C. Brown. Exploring cooperative learning in an introductory computer programming course using visual basic. Master's thesis, University of Missouri-Rolla, Dec. 2005.

[13] C. Brown and R. Pastel. Combining distinct graduate and undergraduate HCI courses: An experiential and interactive approach. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 392–396, New York, NY, USA, 2009. ACM.

[14] C. Brown, R. Pastel, M. Seigel, C. Wallace, and L. Ott. Adding unit test experience to a usability centered project course. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, New York, NY, USA, 2014. ACM.

[15] C. Brown, R. Pastel, B. Siever, and J. Earnest. JUG: A JUnit generation, time complexity analysis and reporting tool to streamline grading. In *Proceedings of the 17th ACM annual conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 99–104, New York, NY, USA, 2012. ACM.

[16] M.-F. Chen. Integrate experiential learning to simulate a website design project process. In *ACM SIGGRAPH 2008 Talks*, SIGGRAPH '08, pages 15:1–15:1, New York, NY, USA, 2008. ACM.

[17] N. Clark. Peer testing in software engineering projects. In *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30*, ACE '04, pages 41–48, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[18] D. Crockford. JavaScript Object Notation (JSON). `http://www.json.org/`, 2006.

[19] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[20] A. M. R. da Cruz and J. ao Pascoal Faria. A metamodel-based approach for automatic user interface generation. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Part I*, MODELS'10, pages 256–270, Berlin, Heidelberg, 2010. Springer-Verlag.

[21] S. H. Edwards. Improving student performance by evaluating how well students test their own programs. *J. Educ. Resour. Comput.*, 3(3), Sept. 2003.

[22] S. H. Edwards and M. A. P.-Q. nones. Experiences using test-driven development with an automated grader. *J. Comput. Small Coll.*, 22:44–50, Jan 2007.

[23] S. H. Edwards and M. A. Perez-Quinones. Web-CAT: Automatically grading programming assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '08, pages 328–328, New York, NY, USA, 2008. ACM.

[24] H. Erdogmus, M. Morisio, and M. Torchiano. On the effectiveness of the test-first approach to programming. *Software Engineering, IEEE Transactions on*, 31(3):226–237, mar 2005.

[25] S. Fraser, K. Beck, B. Caputo, T. Mackinnon, J. Newkirk, and C. Poole. Test driven development (TDD). In *Proceedings of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering*, XP'03, pages 459–462, Berlin, Heidelberg, 2003. Springer-Verlag.

[26] E. F. Gehringer, K. Deibel, J. Hamer, and K. J. Whittington. Cooperative learning: Beyond pair programming and team projects. *SIGCSE Bull.*, 38(1):458–459, 2006.

[27] M. H. Goldwasser. A gimmick to integrate software testing throughout the curriculum. *SIGCSE Bull.*, 34:271–275, Mar 2002.

[28] Google. Android software developer kit. http://developer.android.com/about, 2009.

[29] P. Grigorenko, A. Saabas, and E. Tyugu. Visual tool for generative programming. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-13, pages 249–252, New York, NY, USA, 2005. ACM.

[30] B. Hartfield, T. Winograd, and J. Bennett. Learning HCI design: Mentoring project groups in a course on human-computer interaction. *SIGCSE Bull.*, 24(1):246–251, 1992.

[31] F. Heidenreich, J. Johannes, M. Seifert, C. Wende, and B. Marcel. Generating safe template languages. *SIGPLAN Not.*, 45:99–108, Oct 2009.

[32] J. Hollingsworth. Automatic graders for programming classes. *Commun. ACM*, 3:528–529, Oct 1960.

[33] D. L. Holt, J. Godfrey, and S. Michael. The case against cooperative learning. *Issues in Accounting Education*, 12(1):191–193, 1997.

[34] M. J. Hull, D. Powell, and E. Klein. Infandango: Automated grading for student programming. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 330–330, New York, NY, USA, 2011. ACM.

[35] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 86–93, New York, NY, USA, 2010. ACM.

[36] D. W. Johnson, R. T. Johnson, and K. A.Smith. *Active Learning: Cooperation in the College Classroom*. Interaction Book Company, 1991.

[37] R. R. Johnson. *User-Centered Technology: A Rhetorical Theory for Computers and Other Mundane Artifacts*. State University of New York Press, 1998.

[38] C. Kaner and S. Padmanabhan. Practice and transfer of learning in the teaching of software testing. In *Proceedings of the 20th Conference on Software Engineering Education & Training*, CSEET '07, pages 157–166, Washington, DC, USA, 2007. IEEE Computer Society.

[39] J. Kasurinen, O. Taipale, and K. Smolander. Software test automation in practice: empirical observations. *Adv. Soft. Eng.*, 2010:4:1–4:13, Jan. 2010.

[40] D. G. Kay. Large introductory computer science classes: Strategies for effective course management. *SIGCSE Bull.*, 30:131–134, Mar 1998.

[41] D. G. Kay, T. Scott, P. Isaacson, and K. A. Reek. Automated grading assistance for student programs. In *Proceedings of the Twenty-Fifth SIGCSE Symposium on Computer Science Education*, SIGCSE '94, pages 381–382, New York, NY, USA, 1994. ACM.

[42] D. A. Kolb. *Experiential Learning: Experience as the Source of Learning and Development*. Prentice Hall, Englewood Cliffs, NJ, 1984.

[43] H. Koppelman and B. van Dijk. Creating a realistic context for team projects in HCI. *SIGCSE Bull.*, 38(3):58–62, 2006.

[44] A. F. Kramer and R. M. Schumacher. Laboratory exercises for a graduate/undergraduate course in human-computer interaction. *SIGCHI Bull.*, 21(3):71–75, 1990.

[45] J. Lazar, J. Preece, J. Gasen, and T. Winograd. New issues in teaching HCI: Pinning a tail on a moving donkey. In *CHI '02: CHI '02 Extended Abstracts on Human Factors in Computing Systems*, pages 696–697, New York, NY, USA, 2002. ACM.

[46] B. Lea and C. Brown. A cooperative lecture style and student learning in an introductory computer programming course. *International Journal of Innovation and Learning*, 6(2):192–216, jan 2009.

[47] L. M. Leventhal, J. Barnes, and J. Chao. Term project user interface specifications in a usability engineering course: Challenges and suggestions. In *SIGCSE '04: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, pages 41–45, New York, NY, USA, 2004. ACM.

[48] Y.-K. Lim, A. Pangam, S. Periyasami, and S. Aneja. Comparative analysis of high- and low-fidelity prototypes for more valid usability evaluations of mobile devices. In *NordiCHI '06: Proceedings of the 4th Nordic Conference on Human-Computer Interaction*, pages 291–300, New York, NY, USA, 2006. ACM.

[49] Y.-K. Lim, E. Stolterman, and J. Tenenberg. The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas. *ACM Trans. Comput.-Hum. Interact.*, 15(2):1–27, 2008.

[50] J. Link and P. Frolich. *Unit Testing in Java: How Tests Drive the Code*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[51] Z. Liu, G. Yang, and L. Cai. Test Suite Cooperative Framework on Software Quality. In *Proceedings of the 6th international conference on Cooperative design, visualization, and engineering*, CDVE'09, pages 289–292, Berlin, Heidelberg, 2009. Springer-Verlag.

[52] L. Malmi, V. Karavirta, A. Korhonen, and J. Nikander. Experiences on automatically assessed algorithm simulation exercises with different resubmission policies. *J. Educ. Resour. Comput.*, 5, Sep 2005.

[53] V. Massol and T. Husted. *JUnit in Action*. Manning Publications Co., Greenwich, CT, USA, 2003.

[54] A. Mayer, R. Pastel, C. Wallace, S. Oppliger, and R. Donovan. Environmental cybercitizens: Engaging citizen scientists in global environmental change through crowdsensing and visualization. `http://www.nsf.gov/awardsearch/showAward?AWD_ID=1135523`, 2011.

[55] D. S. McCrickard, C. M. Chewar, and J. Somervell. Design, science, and engineering topics?: Teaching HCI with a unified method. In *SIGCSE '04: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, pages 31–35, New York, NY, USA, 2004. ACM.

[56] M. McCurdy, C. Connors, G. Pyrzak, B. Kanefsky, and A. Vera. Breaking the fidelity barrier: An examination of our current characterization of prototypes and an example of a mixed-fidelity success. In *CHI '06: Proceedings of the SIGCHI Conference on*

*Human Factors in Computing Systems*, pages 1233–1242, New York, NY, USA, 2006. ACM.

[57] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346, July 2002.

[58] MTU. Scientific and Technical Communication Courses. `http://www.mtu.edu/humanities/undergraduate/stc/courses/`, 2013.

[59] I. Newton. *Philosophiae Naturalis Principia Mathematica*. J. Societatis Regiae ac Typis J. Streater, 1687.

[60] J. Nielsen. Heuristic evaluation. pages 25–62, 1994.

[61] R. Pastel. Integrating science and research in a HCI design course. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '05, pages 31–35, New York, NY, USA, 2005. ACM.

[62] R. Pastel. Student assessment of group laboratories in a data structures course. *Journal Computer Science Colliquim*, 22(1):221–230, Oct. 2006.

[63] R. Pastel. Cs 4760. `http://www.csl.mtu.edu/cs4760/www/`, may 2008.

[64] R. Pastel, C. Brown, M. Woller-Carter, and S. Kumar. Teaching human factors to graduate and undergraduate computer science students. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 56, pages 595–599.

[65] J. B. Rainsberger and S. Stirling. *JUnit Recipes: Practical Methods for Programmer Testing*. Manning Publications Co., Greenwich, CT, USA, 2004.

[66] M. Sabin. A collaborative and experiential learning model powered by real-world projects. In *Proceedings of the 9th ACM SIGITE Conference on Information Technology Education*, SIGITE '08, pages 157–164, New York, NY, USA, 2008. ACM.

[67] D. Saff and M. D. Ernst. An experimental evaluation of continuous testing during development. In *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '04, pages 76–85, New York, NY, USA, 2004. ACM.

[68] M. Schlee and J. Vanderdonckt. Generative programming of graphical user interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 403–406, New York, NY, USA, 2004. ACM.

[69] J. Smith, J. Tessler, E. Kramer, and C. Lin. Using peer review to teach software testing. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ICER '12, pages 93–98, New York, NY, USA, 2012. ACM.

[70] S. Thummalapenta, P. Devaki, S. Sinha, S. Chandra, S. Gnanasundaram, D. D. Nagaraj, and S. Sathishkumar. Efficient and change-resilient test automation: an industrial case study. In *Proceedings of the 2013 International Conference on*

*Software Engineering*, ICSE '13, pages 1002–1011, Piscataway, NJ, USA, 2013. IEEE Press.

[71] A. Venables and L. Haywood. Programming students NEED instant feedback! In *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20*, ACE '03, pages 267–272, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[72] J. Wagner, D. Boisvert, J.-P. Kuilboer, J. Keisler, and P. Bharati. Cross-functional concentrations merge IT and business concepts. In *SIGITE '05: Proceedings of the 6th Conference on Information Technology Education*, pages 179–184, New York, NY, USA, 2005. ACM.

[73] K. M. Whicker. *Cooperative Learning in the Secondary Mathematics Classroom*. University of Memphis, 1995.

[74] M. J. Willshire. A usability focus for an HCI project. *J. Comput. Small Coll.*, 17(2):50–58, 2001.

[75] T. Xie, J. de Halleux, N. Tillmann, and W. Schulte. Teaching and training developer-testing techniques and tool support. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, SPLASH '10, pages 175–182, New York, NY, USA, 2010. ACM.

[76] A.-U.-H. Yasar. Enhancing experience prototyping by the help of mixed-fidelity prototypes. In *Mobility '07: Proceedings of the 4th International Conference on Mobile Technology, Applications, and Systems and the 1st International Symposium on Computer Human Interaction in Mobile Technology*, pages 468–473, New York, NY, USA, 2007. ACM.

# Appendix A

# JUG Survey Questions

Students were told to rate each of the following questions based their impact of your learning process.

1. Did having frequent assignments ( 1 / week) help better your understanding?

   Very Unhelpful Unhelpful Moderately Helpful Very Helpful

2. Was the difficulty and quantity of code required helpful in your learning?

   Very Unhelpful Unhelpful Moderately Helpful Very Helpful

3. Did implementing Data Structures (i.e. Sequences, Queues, Maps, Graphs) help you better understand those Data Structures?

   Very Unhelpful Unhelpful Moderately Helpful Very Helpful

4. Did using the Data Structures to Build Algorithms (i.e. Sorting or BeigeChartreusePages) help you better understand those Data Structures?

   Very Unhelpful   Unhelpful   Moderately   Helpful   Very Helpful

5. Did programmatic analysis of your code (to create Time Plots) help you better understand the different time complexities better?

   Very Unhelpful   Unhelpful   Moderately   Helpful   Very Helpful

6. Did you find that programming to an interface and Unit Tests helped you better understand the purpose and use of those Data Structures?

   Very Unhelpful   Unhelpful   Moderately   Helpful   Very Helpful

7. Did annotating and justifying your time and space complexities help you better understand the strengths and weaknesses of different Data Structure implementations?

   Very Unhelpful   Unhelpful   Moderately   Helpful   Very Helpful

8. Are there any aspects of Data Structures that you feel would be good additions to the course?

9. Are there any specific assignments that you feel were particularly unhelpful?

10. Do you have any additional comments concerning the assignments?

The Auto-Grader and Program Reports

Rate each of the following questions based on your perceptions of the grading process.

1. How often did you resubmit your program to attempt to get a better grade?

   Infrequently Not often Sometimes Often Frequently

2. How often did the resubmission improve your grade?

   Infrequently Not often Sometimes Often Frequently

3. How often did you only make a first submission (without penalty)?

   Infrequently Not often Sometimes Often Frequently

4. How often did you only make a second submission (at a -10 penalty)?

   Infrequently Not often Sometimes Often Frequently

5. Did the auto-graded tests match your expectations of the requirements?

   Infrequently Not often Sometimes Often Frequently

6. Did the preliminary reports (the auto-grader) provide clarification on how your code should behave?

   Infrequently Not often Sometimes Often Frequently

7. How much time (in hours) do you think is appropriate for an assignment?

8. About how much time (in hours) do you think you spent on each assignments for this course?

9. How many days would you expect to have between the return of the auto-graded report and the due date for resubmission?
10. How many days would you expect between the resubmission and the hand-graded report?
11. Do you have any additional comments concerning the grade reports?

# Appendix B

# Combined Course Undergraduate Interview Notes

Each interview was scheduled for approximately 30 minutes. The interview was structured with 5 sections, directed by questions (enumerated below). Dr. Robert Pastel and Christopher Brown were present for the entire interview. Student names have been removed for anonymity.

## B.1 Interview Questions

1. What do you think of the pace of the course so far? Why do you feel this way?

2. What aspects of your project has working with the graduate students made you aware of or focused on?

3. How has the "expert" design documentation (the graduate reports) influenced your design process?

4. What limitations has your project experienced as a result of the graduate interaction?

5. Do you have any suggestions for improving the course in the future?

## B.2 Interview Notes

### B.2.1 Portable Beer Pong Scorer

**What do you think of the pace of the course so far? Why do you feel this way?**

The course had a pretty good pace; it wasn't over or underwhelming; everything was realistic

The deadlines were confusing, especially when the graduate students were supposed to have things due.

**What aspects of your project has working with the graduate students made you aware of or focused on?**

The User task analysis with the graduate student helped with scenarios and use cases. The graduate student helped us specify our design.

**How has the "expert" design documentation (the graduate reports) influenced your design process?**

One graduate student’s Heuristic Evaluation had us repeating "remember the simplicity"

**What limitations has your project experienced as a result of the graduate interaction?**

Minor slowdowns from communication.

**Do you have any suggestions for improving the course in the future?**

Get a real device - recommend developer phones for each group about 130 dollars per phone (this group used their own personal phone to program the assignment)

### B.2.2 Groove-on-the-Move

**What do you think of the pace of the course so far? Why do you feel this way?**

"The course was a little fast-paced" There was little time between presentation and review.

**What aspects of your project has working with the graduate students made you aware of or focused on?**

We were forced to re-explain the same thing.

Describing it helped bring it into focus.

**How has the "expert" design documentation (the graduate reports) influenced your design process?**

Provided clarification.

There were 12 action items, including no shuffle/repeat

There were also 5-6 changes added

**What limitations has your project experienced as a result of the graduate interaction?**

Slowed down the process. The Heuristic Evaluation was too late - after the group meeting.

**Do you have any suggestions for improving the course in the future?**

Scheduling and forcing us to read the User Task Goal Analysis and Heuristic Evaluation.

### B.2.3 Location Aware Remote Control (LARC)

**What do you think of the pace of the course so far? Why do you feel this way?**

Steady pace of work assigned

there was a good amount of paperwork

**What aspects of your project has working with the graduate students made you aware of or focused on?**

one graduate student really helped them outside of the work in meetings

helped shape the direction of the project

**How has the "expert" design documentation (the graduate reports) influenced your design process?**

Heuristic Evaluation helped a lot

At this point, the students asked about the final paperwork schedule, deterring from the interview structure.

**What limitations has your project experienced as a result of the graduate interaction?**

made focus better

(This was not a negative comment—they started talking about the limitations of the device. When asked again about limitations caused by graduate interaction, they were confused, so we moved on)

**Do you have any suggestions for improving the course in the future?**

Clarity of the extent of the implementation

Design is not everything

### B.2.4 Mini-Mote

**What do you think of the pace of the course so far? Why do you feel this way?**

The focus of the course was on the design of the interface, there was no reinforcement of reading.

There was no rushing of the meat of the course.

The course was slow during the last two weeks of presentations and interviews.

**What aspects of your project has working with the graduate students made you aware of or focused on?**

We didn't work with one graduate student.

One graduate student provided good information.

The third graduate student did not provide much feedback.

**How has the "expert" design documentation (the graduate reports) influenced your design process?**

“They [the graduates] had good ideas and helped make our project a little better. They were sort of like the customer: they had a unique perspective that we maybe wouldn’t have come up with ourselves.”

The user task goal analysis made us aware that we had a lot and needed to narrow the focus.

**What limitations has your project experienced as a result of the graduate interaction?**

not really - the graduate interaction only really helped

**Do you have any suggestions for improving the course in the future?**

TDA produced a lot of limitations; it is too restrictive.

There should be more leniency or the restrictions more relaxed.

We should choose one of a few project choices for size [scope]

### B.2.5 MP3 Stingray

This interview was quite short - students were not responsive to probing questions.

**What do you think of the pace of the course so far? Why do you feel this way?**

Not rushed and not too much work.

**What aspects of your project has working with the graduate students made you aware of or focused on?**

fleshing out the ideas of how to do it differently

improved upon our design

**How has the "expert" design documentation (the graduate reports) influenced your design process?**

"not much"

**What limitations has your project experienced as a result of the graduate interaction?**

"Not really"

**Do you have any suggestions for improving the course in the future?**

the course was a neat idea

### B.2.6 Run Tracker

**What do you think of the pace of the course so far? Why do you feel this way?**

The pace was good.

The setup was beneficial and the lectures are helping

**What aspects of your project has working with the graduate students made you aware of or focused on?**

One graduate student gave tips to help focus on the user.

With new graduate students, we had to explain the project differently.

**How has the "expert" design documentation (the graduate reports) influenced your design process?**

"It allowed us to push forward once we came up with the task."

Focus on the user.

**What limitations has your project experienced as a result of the graduate interaction?**

"Not really"

**Do you have any suggestions for improving the course in the future?**

More interaction.

Allow time after testing to complete implementation.

### B.2.7 TDA Map

**What do you think of the pace of the course so far? Why do you feel this way?**

The course was too slow.

When able to meet, met for 1:30

The course was too fast when we couldn’t meet

The first few weeks we didn’t do much, it was just presentations and interviews.

Assignments are good.

**What aspects of your project has working with the graduate students made you aware of or focused on?**

Nothing.

Received some positive feedback, such as noting that buttons should be changed.

Also received some negative comments which were unhelpful.

**How has the "expert" design documentation (the graduate reports) influenced your design process?**

The documentation forced us to get things done, but hasn’t made the design better.

**What limitations has your project experienced as a result of the graduate interaction?**

We had delays while waiting for them [graduate students] and them waiting for us.

**Do you have any suggestions for improving the course in the future?**

Less waiting on others

Didn’t know who to work with

less assignments

The course is lacking in material

There should just be testable material

Improve the clarity of grad students assignments

# Appendix C

# Combined Course Graduate Interview Notes

Each interview was scheduled for approximately 30 minutes. The interview was structured with 5 sections, directed by questions (enumerated below). Dr. Robert Pastel and Christopher Brown were present for the entire interview. Student names have been removed for anonymity. One graduate student stopped participating in the course and did not do a final interview.

## C.1 Interview Questions

1. What do you think of the pace of the course so far? Why do you feel this way?

2. Do you feel that the suggestions and documentation have influenced the groups' design decisions?

3. Do you feel that you have been getting adequate information about the projects during the group meetings?

4. Have you experienced any difficulties working with group projects?

5. Do you have any suggestions for improving the course in the future?

## C.2 Interview Notes

### C.2.1 Graduate Student 1

This student took the undergraduate course the previous year.

**What do you think of the pace of the course so far? Why do you feel this way?**

Pace is fine

It seems like the groups met too much with graduate students.

Got a lot out of XWin in 1 lecture than 3 from last year.

Arrange more work for checkpoints (quizzes)

**Do you feel that the suggestions and documentation have influenced the groups' design decisions?**

In general, don't know; iterations are so different its hard to tell what the cuases were

design presentation was a surprise

didn't have much ability to change from association in design phase

**Do you feel that you have been getting adequate information about the projects during the group meetings?**

all of them fleshed out in the first week or two

BeerPong wasn't engaged in the first few design interviews, but going to implement others' implementation will be tough

**Have you experienced any difficulties working with group projects?**

none - groups seem to be having a blast.

Limiting to the TDA encourages creativity

**Do you have any suggestions for improving the course in the future?**

schedule is ambiguous - add more preparation for lecture want to do more (daily) - work toward goal of usability testing

### C.2.2 Graduate Student 2

This student was very interested in the research and was considering expanding on the ideas from the course

**What do you think of the pace of the course so far? Why do you feel this way?**

I feel that it is consistent with the pace of the undergraduates ability to learn

Graduate student schedule is hectic - wait then rush, but not as much as other classes

samples were good - hard to understand some assignments

**Do you feel that the suggestions and documentation have influenced the groups' design decisions?**

yes - looked at mini-mote and stingray: what they were doing was improvement on existing

tapped into experience consistent with existing implementation

recommendation in planning - think about who would benefit

**Do you feel that you have been getting adequate information about the projects during the group meetings?**

I believe that the grads ability to get information depends on the graduate student.

tried to engage in conversation

useful for both

parlay into reflective thought

looked at interaction as leadership rather than graduate student with information

some other grads treated the interviews as "just tell me what I need"

**Have you experienced any difficulties working with group projects?**

Run Tracker - positive and receptive Stingray - not as much back & forth - more defensive

Mini-Mote - lazy and unmotivated - had to halt group talking - unresponsive

**Do you have any suggestions for improving the course in the future?**

lectures could fit more with course

cut XLib

gestalt pictures were fun

read ahead for more participation

graduate student interaction meeting with different groups

not much interest in concept of doing expansion of UG assignments

better fit to paradigm of close interactions biggest problem - participation beat apathy with regular reinforcement

### C.2.3 Graduate Student 3

This student had taken the previous undergraduate HCI course

**What do you think of the pace of the course so far? Why do you feel this way?**

slow - assignments seem to be well paced and don't take too much time

**Do you feel that the suggestions and documentation have influenced the groups' design decisions?**

gives a slightly different take to what they were going to do - tweaked

**Do you feel that you have been getting adequate information about the projects during the group meetings?**

yes - should have diagrams of design storyboard

**Have you experienced any difficulties working with group projects?**

no

**Do you have any suggestions for improving the course in the future?**

no lectures on HCI stuff they would apply to their projects

touchscreen usage

don’t feel involved in class

### C.2.4 Graduate Student 4

**What do you think of the pace of the course so far? Why do you feel this way?**

good (enough time) to slow (more than enough time)

**Do you feel that the suggestions and documentation have influenced the groups' design decisions?**

no, didn't implement

ignored or didn't read documentation

**Do you feel that you have been getting adequate information about the projects during the group meetings?**

for some groups (TDA, UTG, Mini-mote) good information

Groove-on-the-Move did not provide enough information

**Have you experienced any difficulties working with group projects?**

no - it was very good

**Do you have any suggestions for improving the course in the future?**

communication with groups

clear expectations, especially about assignments

### C.2.5 Graduate Student 5

**What do you think of the pace of the course so far? Why do you feel this way?**

its good - lots of interesting information

**Do you feel that the suggestions and documentation have influenced the groups' design decisions?**

not much - first usability goals, not much data

**Do you feel that you have been getting adequate information about the projects during the group meetings?**

yes, Heuristic Evaluation problems with groups didn't understand

**Have you experienced any difficulties working with group projects?**

none

**Do you have any suggestions for improving the course in the future?**

should be more assignments

### C.2.6 Graduate Student 6

**What do you think of the pace of the course so far? Why do you feel this way?**

going smooth - no problems

**Do you feel that the suggestions and documentation have influenced the groups' design decisions?**

yes, helped UTG and LARC

**Do you feel that you have been getting adequate information about the projects during the group meetings?**

yes, from website mainly

**Have you experienced any difficulties working with group projects?**

no

**Do you have any suggestions for improving the course in the future?**

none

# Appendix D

# Unit Test Experience Interview Notes

Each interview was scheduled for approximately 30 minutes and was recorded. A second researcher, asking questions about a different subject, was also present. The group members were assured that the recording was for the use of the researchers only, and would not be distributed to additional parties. A series of questions were asked during the interview to encourage discussion on unit testing through their project. Included are notes of the discussion taken during the interview and after reviewing the recording. Student names have been removed for anonymity.

## D.1 Interview Questions

1. Describe the unit testing you did and how you wrote unit tests.
2. Describe how your unit tests were run, and how often.
3. What types of functionality problems did you encounter during usability testing?
4. What was your biggest obstacle for writing unit tests?
5. What did you learn from writing unit tests?

## D.2 Interview Notes

### D.2.1 Beach

**What do you think of the pace of the course so far? Why do you feel this way?**

This group did not write automated unit tests.

Because of the lack of unit testing, later questions were asked in a different order.

**What was your biggest obstacle for writing unit tests?**

The group had difficulties doing unit testing due to anonymous classes for buttons and text fields.

Their project would generate references "on the fly", which they could not access within a unit test.

**What types of functionality problems did you encounter during usability testing?**

Email field didn't do anything - data was not passed correctly

GPS entry didn't work

There was code to do the GPS entry, but it didn't work in conjunction with the back button

The program lagged when scrolling through list of data.

**Describe how your unit tests were run, and how often.**

: The group tested their program by running through process regularly throughout development.

They needed to use a real phone because emulator couldn't send e-mails.

Their tests involved setting up beaches with a large number of variables, enter them and send as e-mail.

They would try to break the application. For instance, they would enter a beach, delete it, then create another with the same name.

Constantly through development, when making a change, they would try to find ways to break it.

**What did you learn from writing unit tests?**

: Emulator takes quite a while to load.

With dynamic xml testing, they recognized that a new project with knowledge of unit testing ahead of time, they would ensure a way to access that data.

They barely finished their project in time.

Dynamic generation is very difficult - Pastel said it would be hard, but they shrugged off - turned out to be hard.

### D.2.2 Lichen

**What do you think of the pace of the course so far? Why do you feel this way?**

Requested Filtration system - was "axed".

Scientist cut number of lichens so that filtration was unnecessary.

Different aspect.

"play" with unit test.

one unit test

compound component which had to inflate

unit tests to ensure that it received the proper input

**Describe how your unit tests were run, and how often.**

created the component, generated the values expected.

ran using eclipse

frequency: just during development of that page.

ran 3 times over the course of 1 day.

**What types of functionality problems did you encounter during usability testing?**

errors in implementation

bugs - sensitivity of slider bar

two applications - one on morph page, one on species

species page

large list - scroll down and scroll up, input is gone and data underneath would randomize

back button didn't go back - moved to whatever was selected

activity would crash and bring you to previous page

return to start crashed

**What was your biggest obstacle for writing unit tests?**

not familiar with unit tests

courses that familiarized with unit testing

knowing what to unit test

could not unit test - buttons forward/back, unit GPS (lock on) plans to unit test list view/adapter

**What did you learn from writing unit tests?**

handy - way eclipse - passes data to tests

it could handle unit tests without launching emulator, which is slow and clunky

**Extra Discussion**

Due to some extra time waiting for the other interviewer, we chatted a little more about unit testing)

Students asked about interface objects that require checking a click event (from response to question 4), how do you test it?

Interviewer explained that for a click event or other required action, unit tests should fake the event. First, the test should load the initial page, send the event, then see what the state is next. Unit tests can be broken down to a lot of different levels, but essentially, a unit test is designed to create a state as setup, do something, then check the system state afterward.

Students asked about using a unit test to "fake" GPS coordinates. Interviewer did not know technical details about unit testing with an Android's GPS.

Students asked about unit testing the layout of an activity. Interviewer explained that this is not possible, because there must be a visual inspection based on the intent of the designer. In the same way, usability tests are dissimilar to unit tests, since the latter can only test specific functionality, not general usage.

### D.2.3 ROV

This group did not participate in usability testing, and did not "test" their program due to inadequate support from their scientist (customer). Therefore, Questions 2–4 were skipped, and question 5 referred to previous experiences with unit testing (including the classroom exercise).

**What do you think of the pace of the course so far? Why do you feel this way?**

Group wasn't able to implement any unit testing - requested testing was on signals sent to motor.
Group wasn't able to hook up to ROV.

Group didn't write unit tests for anything.

The project consists of one main file.

The main program relied on video coming from a laptop that didn't have access to.

They were to generate signals to send to ROV, but had no access to the ROV unit.

They did not do usability test for same reasons.

**What did you learn from writing unit tests?**

Some of the group members had done unit testing before in other courses, including: Software Quality Assurance and Team Software Project.

When asked about the experience earlier in the course, none of the group members remembered the unit test assignment.

### D.2.4 Stream Features

**What do you think of the pace of the course so far? Why do you feel this way?**

This group wrote JUnit tests for their database.

This was the largest piece of their project.

They were asked not to do unit testing on their listview, but that aspect of the program wasn’t included in the final product.

When asked about process, one student said they wrote all the tests, as the "only one who knew anything about databases."
At the time of writing tests, that student wasn’t sure about the other Activities being designed for the application. However, they did know what Database function calls would be, and that it needed to be able to delete rows and create rows on various tables. The program also needed to recursively create and delete rows in response to modifications in other tables.

**Describe how your unit tests were run, and how often.**

The group used JUnit through Eclipse.

They ran the code initially without unit tests to ensure that it was connecting correctly.

They made the Database class, and ran unit tests by calling methods directly.

As activites were put in, those activities would call Database functions indirectly.

Unit tests were run during development as various methods were used.

**What types of functionality problems did you encounter during usability testing?**

no functionality problems - cosmetic problems filed bug reports based on layout designs the DB is pretty much the entire project

**What was your biggest obstacle for writing unit tests?**

relearning JUnit done in a previous class

only a few test cases failed initially - fixed

classes that you've seen unit testing? Team Software Project

**What did you learn from writing unit tests?**

really useful for testing SQL-like interfaces, since difficult to look at rows of DB feel like sometimes JUnit testing is a lot of work that doesn't pay off other times if you set it up correctly / something that needs test cases

### D.2.5 Tracker

*? This group wrote their tests after the usability testing

**What do you think of the pace of the course so far? Why do you feel this way?**

Text export of routes, data going in matched data coming out of file

wrote based on expandable list object. used generics with export activity scanned output file - ensured that every output was the same as input

**Describe how your unit tests were run, and how often.**

eclipse to run

just a couple times at the end

**What types of functionality problems did you encounter during usability testing?**

2 menus that show up on map. if both were open, they would overlap

delete a marker wouldn't close marker dialog - had to click twice, then it would crash program taking pictures added a second marker instead of adding to existing marker weren't getting source correctly for pictures

**What was your biggest obstacle for writing unit tests?**

familiarizing with unit testing (one person wrote all of them)

**What did you learn from writing unit tests?**

didn't learn a lot about the App, just about writing unit tests

learned more about unit testing through lecture

supposed to test kmz exporting, but unable to implement

# Appendix E

# Unit Test Experience Survey Questions

## CS4760 Class Testing Survey - Spring 2013

The purpose of this survey is to learn more about your experiences with combining Unit Testing and Usability Testing. Please read each question carefully and answer them to the best of your ability. Do not take this survey until usability testing of your app is complete. During this survey, the following terms will be used:

† **Activities**: Single pages on the Android device which contribute to the main App.

† **Usability Testing**: Tests intended to identify usability issues in a user interface. This primarily refers to the end-of-semester tests that were run by the graduate students in

CS 5760.

- † **Unit Testing**: Any repeatable, self-contained test which verifies specific functionality within a program.

- † **Test Driven Development (TDD)**: The practice of writing unit tests before implementing the functionality that the tests are intended to verify.

## Testing Practices prior to CS 4760

The questions in this section ask about projects you have worked on before taking CS 4760. You may recognize situations where you used testing practices without knowing the terminology involved. Consider these situations when answering the questions, and please do not consider projects that you worked on after beginning CS 4760.

**Before participating in CS 4760, I have ________ participated in projects that used Unit Testing practices.**

1. never
2. seldom
3. occasionally
4. frequently

5. regularly

**Before participating in CS 4760, I personally ____ write unit tests for projects.**

1. never
2. seldom
3. occasionally
4. frequently
5. regularly

**Before participating in CS 4760, I have ______ used Test Driven Development (TDD) practices in projects (where unit tests were written before the functionality was implemented).**

1. never
2. seldom
3. occasionally
4. frequently
5. regularly

## Unit Testing during CS 4760

The questions and statements in this section ask about specific aspects of Unit Testing within the CS4760 HCI course. Please address your answers with respect to that specific part of the course.

**Which app did your group make?**

† Beach

† Lichen

† ROV

† Stream Features

† Tracker

**Were you responsible for writing Unit Tests for any activities?**

† Yes

† No

**What portion of your group's project did you personally write unit tests for?**

1. none
2. a few
3. half
4. most
5. all

**Which Activities did you personally write a significant amount of code for?**

**Which Activities were not unit tested?**

**Approximately how many unit test methods did your group write for each Activity?**

† none

† between 1 to 5

† between 6 to 10

† between 11 to 15

† between 16 to 20

† between 21 to 30

† between 31 to 40

† between 41 to 50

† more than 50

**What portion of your group's unit tests were written before the code (Test Driven Development)?**

1. none
2. a few
3. half
4. most
5. all

## Bugs and Errors Discovered

The questions in this section refer to functionality problems discovered after the Final Design Presentation, when the App is intended to be completely functional.

**How many functionality problems were discovered before Usability Testing?**

† none

† between 1 to 5

† between 6 to 10

† between 11 to 15

† between 16 to 20

† between 21 to 30

† between 31 to 40

† between 41 to 50

† more than 50

**How many functionality problems were anticipated and fixed while writing Unit Tests?**

† none

† between 1 to 5

† between 6 to 10

† between 11 to 15

† between 16 to 20

† between 21 to 30

† between 31 to 40

† between 41 to 50

† more than 50

**How many functionality problems were discovered during Usability Testing?**

† none

† between 1 to 5

† between 6 to 10

† between 11 to 15

† between 16 to 20

† between 21 to 30

† between 31 to 40

† between 41 to 50

† more than 50

**What portion of the functionality problems were discovered in Activities that had Unit Tests?**

1. none
2. a few
3. half
4. most
5. all

**What portion of functionality problems were discovered in Activities that did not have unit tests?**

1. none
2. a few
3. half
4. most
5. all

## Unit Tests vs Usability Tests

The following questions are to provide us with information on identifying the type of testing to use in various situations. In these scenarios, a Unit Test is an automated program which validates the functionality of a program. A Usability Test presents a paid test subject with a scenario to evaluate usability concerns. [1]

For these questions, imagine that you are contracted to create a website for a local business that is building an online store to sell âĂIJknickknacksâĂİ. You have already negotiated a price for the following items, with the understanding that you demonstrate that you have completed each item.

**On the main page there should be a set of featured product images which rotate every hour or so. Clicking on the image should link to that products information page.**

† *Unit Tests*

† Usability Tests

† Both Unit and Usability Tests

† Neither Unit Tests or Usability Tests

**After a user logs in, they should be able to find purchase and tracking information about a previous order that has already been filled.**

† Unit Tests

[1]Expected answers are in *italics*, but were not in the original survey

† *Usability Tests*

† Both Unit and Usability Tests

† Neither Unit Tests or Usability Tests

**Some of the gift items have custom text for them, specifically names and addresses of the recipient. You are to have these text boxes propose Autofill suggestions with common American names and streets.**

† Unit Tests

† Usability Tests

† *Both Unit and Usability Tests*

† Neither Unit Tests or Usability Tests

**You are given a paper brochure with descriptions of each of the products (these were originally mailed to previous customers). The descriptions on the website for each product should match the brochure's description.**

† Unit Tests

† Usability Tests

† Both Unit and Usability Tests

† *Neither Unit Tests or Usability Tests*

**Since many of the items are non-traditional, it is difficult to categorize items. Therefore, you have been asked to dynamically categorize items based on search terms and purchase habits of customers. These categories should be provided as links to groups of items, and be moved based on the frequency they are used.**

† Unit Tests

† Usability Tests

† *Both Unit and Usability Tests*

† Neither Unit Tests or Usability Tests

## General Testing

These questions are about your current opinions on unit testing and usability testing. They also inquire about your expectations for future projects.

**How necessary is Unit Testing to software development?**

† A poor use of limited time

† An optional activity

† An activity required for only large projects

† An activity required for all projects

† A necessary step in all levels of development

**How necessary is Usability Testing to user interface development?**

† A poor use of limited time

† An optional activity

† An activity required for only large projects

† An activity required for all projects

† A necessary step in all levels of development

**How necessary is Test Driven Development to software development?**

† A poor use of limited time

† An optional activity

† An activity required for only large projects

† An activity required for all projects

† A necessary step in all levels of development

**I will use Usability Testing when developing personal projects.**

Please indicate your agreement to the above statement, where 1 means strongly disagree, 2 disagree, 3 neutral, 4 agree, and 5 strongly agree.

1. strongly disagree
2. disagree
3. neutral
4. agree
5. strongly agree

**I will encourage the use of Usability Testing when developing professional projects.**

Please indicate your agreement to the above statement, where 1 means strongly disagree, 2 disagree, 3 neutral, 4 agree, and 5 strongly agree.

1. strongly disagree

2. disagree
3. neutral
4. agree
5. strongly agree

**I will use Unit Testing when developing personal projects.**

Please indicate your agreement to the above statement, where 1 means strongly disagree, 2 disagree, 3 neutral, 4 agree, and 5 strongly agree.

1. strongly disagree
2. disagree
3. neutral
4. agree
5. strongly agree

**I will encourage the use of Unit Testing when developing professional projects.**

Please indicate your agreement to the above statement, where 1 means strongly disagree, 2 disagree, 3 neutral, 4 agree, and 5 strongly agree.

1. strongly disagree
2. disagree
3. neutral
4. agree
5. strongly agree

**I will use Test Driven Development when developing personal projects.**

Please indicate your agreement to the above statement, where 1 means strongly disagree, 2 disagree, 3 neutral, 4 agree, and 5 strongly agree.

1. strongly disagree
2. disagree
3. neutral
4. agree
5. strongly agree

**I will encourage the use of Test Driven Development when developing professional projects.**

Please indicate your agreement to the above statement, where 1 means strongly disagree, 2 disagree, 3 neutral, 4 agree, and 5 strongly agree.

1. strongly disagree
2. disagree
3. neutral
4. agree
5. strongly agree

## Open Ended Questions

Please write about specific concepts, examples and your feelings that apply for each question. I recommend that you compose your answers in a separate document and then paste them into the text field for each question.

**Please write about any of your experiences where unit testing helped to improve the code.**

**What were the most notable software bugs discovered during Usability Testing, and how were they handled?**

**Was there any specific code that you thought should be unit tested and was not? Why should it have been unit tested? Why wasn't it?**

**Do you have any additional comments or thoughts on Unit Testing or Test Driven Development?**

# Appendix F

# Unit Test Experience Bug Reports

Note that since there were more graduate students than teams, the "Beach" application had two graduate students assigned to do usability testing. In addition, each graduate student was responsible for collecting bug reports and formatting them for their final report. The reporting of the following bugs have been slightly modified from those original reports. Names and e-mails have been removed.

## F.1 Beach (Graduate Student 1)

Notes recorded by *[one of the developers]*

Auto get gps button does not work.

Gps doesn’t update on some phones until you hide the virtual keyboard.

*Data got reset after adding another variable.* Data entry $\rightarrow$ Back $\rightarrow$ Add variable $\rightarrow$ Save & continue $\rightarrow$ *data is reset*

*From edit beach, save and continue button kill data* (save and continue button kills data)

Drifter: Check save and return

Drifter: Not calling update on back from

New beach $\rightarrow$ Save and continue $\rightarrow$ Back $\rightarrow$ Back

Drifter: proposed fix: Update beach list on back button from edit beach screen or Update beach list on save and continue press

Scrolling to air temp, some lag issues

e-mail field is useless right now

Bug: Bouncing Submit button on the data record screen

Bug: GPS data wasn’t being saved when moving around screens

1 feet

*Worry over data saving**need to allay fears*

GPS doesn’t update text on empty ok

Led to believe can email from home screen since text entry is there

*EMERGENCY* GPS data still isn’t saving... on the use case where they change beaches and come back.

*It is a bit slow and laggy*

## F.2 Beach (Graduate Student 2)

Scenario: ___all____

Date & Time: ____17

th April_____

1. Describe how this bug/problem occurred:

Every time when user enter the variable page , it will go time bar and pop out keyboard , user need to cancel it every time when entering this page.

2. Repeat the same operation , is bug still happen(if it is potential bug)?

Yes , it always happen

3. What is the result this bug/problem lead to ?

Just not convenience

4.The feedback of this bug from app (if it has)Bug

Scenario: ___B&C____

Date & Time: ____19

th April_____

4. Describe how this bug/problem occurred:

App accident quit after screen unlock in variable page

5. Repeat the same operation , is bug still happen(if it is potential bug)?

Not happen .

6. What is the result this bug/problem lead to ?

App stop and quit

4.The feedback of this bug from app (if it has)

Sorry , beach app accidently stop

## F.3 Lichen

Bug Description

Activity When Bug Occurred

Error Message Actions Taken

Frequency Of Occurrence During Testing

Return to beginning

button crashes activity

Finish Site Screen (Test Air Quality)

Unfortunately, Lichen AQ has stopped

Had users stop before this task

1

| | | | | |
|---|---|---|---|---|
| White text on white background | New Detailed Site | None | Had users bypass this activity | 1 |
| Next button fails to do anything | New Simple Site | None | Avoided simple site | 1 |
| Data disappears when scrolling | Detail Site | None | Had users manually record rating on paper | 1 |
| Wrong image displayed when enlarged | Detail Site | None | None | 3 |

## F.4 ROV

The ROV group did not participate in usability testing and the associated graduate student did not file bug reports.

## F.5 Stream Features

Report 1.

Date & Time: Apr. 15 2013 13:12

Description of the bug/problem occurring:

Other selection on ____ screen for filling in data is missing free form data entry.

Activities at the time of the problem:

Selecting other.

Report 2. Date & Time: 4-15-13 3.20 pm

Description of the bug/problem occurring:

Focus on some views

Activities at the time of the problem:

Multiple data entry activities (erosion + treatment

## F.6 Tracker

id

Bug Description

Date Filed

Status

001

Application sometimes crashes when a marker is deleted

04/17/13

Open

002

The edit menu will display over the new marker drawer

04/17/13

Resolved

003

The built in settings button is not functional

04/17/13

Open

004

Exiting the application clears data

04/17/13

Open

005

Tutorial persists even if user selects âĂIJNoâĂİ

04/18/13

Open