AN INVESTIGATION OF THE IMPACT OF AUTOMATED SOFTWARE TESTING TOOLS ON REFLECTIVE THINKING AND STUDENT PERFORMANCE IN INTRODUCTORY COMPUTER SCIENCE PROGRAMMING ASSIGNMENTS

by

Evorell Lawton Fridge

M.A., Louisiana State University, 2003

B.G.S., University of Louisiana - Lafayette, 2001

A dissertation submitted to the Department of Research and Advanced Studies
College of Professional Studies
The University of West Florida
In partial fulfillment of the requirements for the degree of
Doctor of Education

2014

The dissertation of Evorell Lawton Fridge is approved:

Thomas J. Kramer, Ph.D., Committee Member	Date
John W. Coffey, Ed.D., Committee Member	Date
Sikha S. Bagui, Ed.D., Committee Chair	Date

Accepted for the Department/Division:

Patricia C. Wentz, Ph.D., Chair	Date

Accepted for the University:

Richard S. Podemski, Ph.D., Dean, Graduate School	Date

ACKNOWLEDGMENTS

Thank you to Dr. Bagui for agreeing to work with me on this long dissertation process, and for continuing to honor that commitment even when the circumstances of life would have made it easy for you to bow out. Your strength and work ethic are very inspiring. Thank you to Dr. Coffey for all the advice and encouragement, and to Dr. Kramer for providing valuable feedback and serving on this committee even after retirement.

To my family, I appreciate all the sacrifices you've made to help me finish what I started. I would like to thank my father for inspiring me to pursue my doctorate. I also want to thank my wife for her never-ending patience and encouragement throughout this process. To my sons: you can achieve anything you set your mind to.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

AN INVESTIGATION OF THE IMPACT OF AUTOMATED SOFTWARE TESTING TOOLS ON REFLECTIVE THINKING AND STUDENT PERFORMANCE IN INTRODUCTORY COMPUTER SCIENCE PROGRAMMING ASSIGNMENTS

Evorell Lawton Fridge

This research examined the benefits of automated software testing on student performance and levels of reflection. Edwards (2004) theorized that the increased grade performance that he observed in students who used his Web Center for Automated Testing (Web-CAT) software was the result of increased levels reflective thought in students, such as the reflection-in-action described by Schön (1983). The participants in this study consisted of 144 students in introductory Java programming courses at the University of West Florida. Students were invited to use the Web-CAT software-testing tool for three software projects in the middle of a semester. Students were not required to write their own test cases. Instead, the testing tool used researcher-supplied test cases to evaluate student code and provide immediate feedback to the students. At the end of the semester, student self-reported levels of reflection were measured using Kember et al.'s (2000) reflective thinking survey. Students who used the software were grouped into three usage levels: none, low, and high. The only significant difference in the levels of reflective thought among any of the usage categories was a lower level of reported understanding for the high Web-CAT usage level. Average student project performance also increased significantly for those in the high usage level. Students, instructors, and administrators could benefit from the adoption of such automated testing software and may see improvements in student performance even without student-written test cases. More research is needed, however, to determine if student-written test cases would provide an increase in student reflective thought.

CHAPTER I

INTRODUCTION

Learning to write good computer code is a difficult task. Students can sit in a classroom and learn what constitutes good programming practice, but this is a very different thing from actual programming experience. This is why programming assignments are often used in computer science, even in introductory programming courses. This distinction between the theory of classroom work and the experiential learning that happens in programming assignments is highlighted even more by the individual nature of the assignments. While in-person classroom experiences encourage participation and discussion, instructors will often assign programming tasks to students as solo work with strict warnings against collaboration. It can be all too easy for a student in an introductory course to excel at theoretical assignments like tests and quizzes and yet struggle through their programming assignments without asking for assistance.

Many instructors have observed the tendency for students to use a “Brownian motion” approach to programming that involves small random changes to code in the hopes that some solution would eventually arise (Edwards, 2004; Reek, 1989; Spacco, 2006). This strategy is named after the random way particles travel in a fluid. Students who take this approach make adjustments to their program without a particular plan or direction, frantically hoping to get their program to do something useful. They continue to hit the compile button and hope that whatever they just did makes their code work. Another popular approach that novices take is “Big Bang” coding (Edwards, 2003), which involves writing large amounts of computer code at one sitting without any testing at all. This strategy is named after the scientific theory that the universe was created in one sudden event. Students who use this method do not take an iterative approach to development, but rather write all their code out at once. They are often disappointed to learn that their program does not work as expected when they finally run it, and it is then difficult for them to pinpoint their problems.

Once a student's assignment is finally completed he or she must submit it to their instructor and wait to hear back about it. Depending on the size of a computer science class, students may have to wait a week or two from the time an assignment is due until the time that they receive feedback on their performance. A recent survey of computer science instructors indicated that an average of 15 minutes was spent grading each assignment, usually with a checklist or a set of predetermined test cases (Spacco, 2006). By the time an assignment is returned, a student may have already been tested on the principles emphasized in the assignment or the focus of the class might have shifted to another topic. It can be difficult for students to integrate the feedback they have received and improve their knowledge of a subject when so much time has passed.

Background to the Study

There is no substitute for sitting in front of a computer screen and working through a problem. In this respect computer science has much in common with the art and performance-oriented disciplines of architecture and design. In these more artistic schools of learning, instructors often come alongside students and work with them one on one to discover weaknesses in their technique and areas for improvement. These disciplines feature experiences called practicums because, like programming assignments in computer science, they are designed to give students a taste of real life practice in a controlled setting.

Schön (1987) observed the importance of reflection in these practicums when he described the interaction between a teacher and his student in an architecture studio. The teacher's guidance took the form of a "reflective conversation" (Schön, 1987, p. 56) with the student's problematic situation, moving things around on the sketchpad and continuing to try new experiments until a workable solution was found. This dialogue allowed the instructor to model a proper design process while also teaching the student how to think like a designer. Schön (1983) coined the term *reflection-in-action* to describe this process of encountering a problem, reflecting on possible causes and solutions while still engaged in the situation, and conducting an experiment to test the possible solution for effectiveness. Schön (1983) has credited the ability of certain talented professionals to think on their feet as a reason for their success.

This kind of one-on-one interaction between a student and teacher certainly has the potential to help students overcome problems that they facing, yet computer science educators rarely have the time to sit down with each of their students to assist them with all of their assignments. Edwards (2003) and Spacco (2006) have proposed the use of automated software-testing systems and an increased emphasis on student-written test cases as a means of improving student performance on programming assignments. Because these tests can show where their program deviates from a specification, automated testing tools can provide feedback that is similar to what the students would receive from instructors or clients.

Automated software-testing systems provide a certain measure of feedback to students by reporting the results of a series of test cases immediately. They also provide a benefit to instructors by automating the monotonous task of testing for validity so that the instructors may instead focus on grading for quality. Several institutions have used versions of this sort of tool in the past with some measure of success (Douce, Livingstone, & Orwell, 2005), but they have not come anywhere close to universal adoption. Automated software-testing tools have been criticized in the past for encouraging students to "focus on output correctness" (Edwards, 2004, p. 28) at the expense of proper design and testing. Additional challenges to widespread adoption of these graders are the need to design programming assignments to work with automated graders and the added overhead and expertise needed to run these systems (Spacco, 2006).

Recently, a new class of automated software-testing systems called *meta-graders* has been developed to allow students to submit their own test cases for evaluation (Edwards, 2003; Spacco, 2006). These tools are based on the test-driven development (TDD; Beck, 2003) approach to coding. They allow students to write their own test cases and submit them along with their computer program to a web-based program for analysis. The programs analyze the student code and provide them with instantaneous feedback using the tests provided by the student as well as a test suite written by the instructor. Edwards' study (2004) indicated that students who participated in test-driven design using his new meta-grader had an increase in grade performance over those who used an earlier, more primitive automated software-testing tool that simply reported a grade without any suggestions for improvement.

Statement of the Problem

Previous research into the effectiveness of TDD has shown that it contributes very little on its own to improvements in programmer productivity or the quality of the software that he or she produces (Kollanus, 2010). The use of TDD was, however, associated with an increased level of time and thought spent on the development and testing of software (Huang & Holcombe, 2008; Marrero & Settle, 2005). Edwards (2004) associated the use of student test cases in automated software testing with fostering an environment of on the spot experimentation that is so closely associated with reflection-in-action. However, Edwards (2004) did not attempt to measure whether or not an increase in reflection was actually occurring, nor did he attempt to link this measurement to student performance. Further research is needed to determine whether or not an increase in reflective thought can be observed when automated software testing is introduced, and whether or not this can be linked with any improvement in student performance. Perhaps the environment of reflection theorized by Edwards (2004) has less to do with the actual mechanics of TDD and more to do with the use of software tools that encourage testing and give more immediate feedback than a compiler can alone.

Statement of Purpose

The purpose of this study was to examine the effectiveness of an automated software-testing environment on the average project grade performance of students in an introductory computer science class. It also examined any influence that reflective thought may have on student performance. It was hypothesized that students would have higher average project scores and increased levels of reflective thought when using an automated software-testing environment.

Research Questions

This study used six research questions to investigate the relationships between automated software testing, reflection and student performance. The first three questions investigated the central ideas of this study, while the remaining three investigated the influence that demographics might have on any of these variables.

Research Question 1. How does the use of automated software testing influence levels of reflective thought in students compared to students who do not use automated software testing?

Research Question 2. How does the use of automated software testing influence student performance on introductory computer science programming compared to students who do not use automated software testing?

Research Question 3. To what degree does reflective thought affect student performance on programming assignments for those who use automated software testing compared to those who do not?

Research Question 4. How does demographic data influence levels of reflective thought in students both who use and who do not use automated software testing?

Research Question 5. How does demographic data influence student performance on introductory computer science programming assignments?

Research Question 6. How does demographic data influence student usage of an automated software-testing environment?

Significance of the Study

This study adds to the literature of computer science education by exploring the relationship between reflection and automated software testing first theorized by Edwards (2004). It explores the influence of such tools on student performance, but it deviates from Edwards' (2004) design by offering students researcher-provided tests instead of asking them to write their own. It then examines the role that reflection might play in any observed improvement in student performance.

An investigation into the effect of automated software testing on student performance and levels of reflective thought will allow this technology to be judged by its ability to promote student learning. This study will help administrators and faculty in computer science departments decide whether it is worth the extra time and effort required to deploy and maintain automated testing software in their departments. Instructors could rely on their test cases to grade student assignments against a given set of requirements and focus more of their time on evaluating the student's design and coding style. Students would get the added benefits of an environment that provides instantaneous feedback and encourages a purposeful approach to software development.

Key Terms

Critical Reflection. Critical reflection is a high level of reflection that can "cause us to be critical of epistemic, social, or psychological presuppositions" (Mezirow, 1991, p. 108). Critical reflection can result in a change of perspective or philosophy when approaching a problem.

Experiential Learning. Experiential Learning is a theory of learning that emphasizes the role that experience plays in the learning process. It is defined as "the process whereby knowledge is created through the transformation of experience" (Kolb, 1984, p. 38).

Reflection (or Reflective Thought). Reflection is defined as "active, persistent, and careful consideration of any belief or supposed form of knowledge in the light of the grounds that support it" (Dewey, 1910, p. 6). Reflection is measured by an instrument of reflective thought (Kember et al., 2000) containing four scales: habitual action, understanding, reflection, and critical reflection.

Student Age. Student age is the self-reported age of the student at the time of the study.

Student Classification. Student classification is the self-reported classification of the student at the time of the study (e.g., Freshman, Sophomore, Junior, Senior, Graduate Student).

Student Major. Student major is the self-reported major of the student at the time of the study (e.g., computer science, math, electrical engineering, etc.)

Student Performance. Student performance will be calculated as the average of three of a student's programming assignments (projects 3 through 5) for the semester.

Test-Driven Development. Test-driven development is an approach to software development where test cases are written before any software is written (Beck, 2003). These test cases are used as a means of design and verification of software.

Organization of this Dissertation

This dissertation is separated into five chapters. The first chapter serves as an introduction to the problem and research questions being investigated. The second chapter is a review of the literature surrounding reflection and automated testing. The third chapter describes the methods used in testing the hypotheses, while the fourth chapter presents the results obtained through these methods and the statistical analysis of those results. The fifth chapter contains the conclusions of the researcher and the implications of the results, along with the limitations of the study and suggestions for further research.

CHAPTER II

LITERATURE REVIEW

Introductory computer science classes typically use programming assignments in addition to traditional classroom lectures. These assignments introduce opportunities for students to apply what they have learned in a practical way. This *experiential learning* (Kolb, 1984) approach to education is popular in areas where learning is measured not just by whether or not the students understand the material, but whether they can also do the work asked of them. In this respect, programming assignments have much in common with creative design processes that are "learnable, coachable, but not teachable" (Schön, 1987, p. 157).

Theoretical Basis

In his theory of reflection-in-action, Schön (1983) described a way in which students might actively engage with and learn from their interactive experiences. In a broader sense, this type of purposeful thought is related to experiential learning (Dewey, 1938; Kolb, 1984), which in turn is based on the constructivist approach to learning first introduced by Piaget (1928).

The theory of constructivism includes the formation and adaptation of cognitive structures that are then tested and adapted as a person encounters different life experiences. These cognitive structures are the viewpoint from which we observe our world and make sense of what is happening around us. From a constructivist point of view, reflection-in-action is the act of forming cognitive structures based on our past professional experience, then testing these structures in practice. If a weakness is found in them, an on-the-spot experiment is then conducted to come up with a new cognitive structure that allows us to better approach the problem. As Schön (1987) has said, "in the constructivist view, our perceptions, appreciations, and beliefs are rooted in worlds of our own making that we come to accept as reality" (p. 36).

The first part of this chapter contains a review of the occurrence of reflection in learning theories. There is also a discussion of Schön's concept of reflection-in-action and the use of reflection in an educational setting, as well as some examples of the application of these theories in research. The second part of this chapter involves a review of the literature of automated grading in the context of computer science education. It also contains a discussion of the effectiveness of TDD and the importance of reflection in learning to program a computer.

Reflection

Since Schön (1983) first emphasized the role of reflection in professional practice in his book *The Reflective Practitioner*, the topic of reflection has become one of interest for educators in many professional disciplines (Hatton & Smith, 1995). Reflection is an integral part of the learning process, so much of the interest in reflection hinges on its role in learning and the degree to which it can enhance the learning process. In order to understand Schön's theories of reflective practice and reflection-in-action, it is first useful to explore the role that reflection has played in educational research and learning theory.

Reflection in learning theory. Reflection upon experiences tends to improve our understanding and ability to learn from them. For this reason, the topic of reflection seems to occur most often in the literature of experiential education. Researchers like Dewey (1938), Lewin (1946), and Kolb (1984) have all included the concept of reflection in their theories of how learning occurs.

Dewey (1938) described a cycle of experiential learning in his book *Experience and Education*. He described the process of "formation of purposes" as an observation, followed by an application of knowledge and judgment upon a situation. These judgments can become the observations that we see in successive cycles. Dewey (1938) contrasted the formation of a purpose with acting on an impulse, saying "the crucial educational problem is … postponement of immediate action upon desire until observation and judgment have intervened" (p. 69). Implicit in the phases of observation and the application of previous knowledge is the idea that the learner will reflect on things that he or she has learned in the past and apply that knowledge to future actions.

Dewey (1910) also discussed the concept of reflection at length in his work *How We Think*, in which he defined reflection as "active, persistent, and careful consideration of any belief or supposed form of knowledge in the light of the grounds that support it, and the further conclusions to which it tends" (p. 6). He also spoke of two steps in every reflective operation. A "state of perplexity, hesitation [and] doubt" is always accompanied by "an act of search or investigation directed toward bringing to light further facts which serve to corroborate or nullify the suggested belief" (Dewey, 1910, p. 9). These steps hint at the reflective cycles that would be developed by future researchers like Lewin (1946), Kolb (1984) and Schön (1983).

Dewey (1910) spoke of reflection as a passive, logical activity. He said that "reflection is turning a topic over in various aspects and in various lights," and that the terms "weigh, ponder [and] deliberate" are closely associated with this activity (p. 57). He also spoke of the importance of consulting past experiences when deciding future actions. This pensive approach to reflection is similar to that described by Boud, Keogh, and Walker (1985), in contrast with Schön's immediate reflection-in-action in the middle of an experience.

Lewin (1946) placed great emphasis on the integration of practical and theoretical research, and his model of action research introduced the concept of the *feedback loop* to tie together these two worlds. The loop starts with a concrete experience and then proceeds through stages of observation and reflection, formation of abstract concepts, and finally testing those concepts in new situations (Lewin, 1946). Kolb (1984) has credited Lewin's "continuous process of goal-directed action" (pp. 21-22) as the inspiration for many of his ideas on experiential learning.

Kolb's (1984) work on learning styles also used a cyclical pattern. Kolb (1984) identified four modes of experiential learning that can be viewed as a cycle representing the learning process: concrete experience, reflective observation, abstract conceptualization, and active experimentation. These dimensions can also be viewed as two scales of measurement perpendicular to one another, similar to an X and Y-axis on a graph. Concrete experience contrasts with abstract conceptualization, and reflective observation contrasts with active experimentation. These dimensions form the basis of his Learning Style Inventory (LSI), which provided a way to measure the learning strengths of individuals on this scale (Kolb, 1984).

The dimension of Kolb's (1984) scale called *reflective observation* is of particular importance to this study. The learner who is strong in this area best understands a concept when given time alone to think about it apart from any other activity. Reflection alone, however, will not have the desired impact on learning. Instead, Kolb said that "learners, if they are to be effective need four different kinds of abilities" and that the learner "must continually choose which set of learning abilities he or she will bring to bear in any specific learning situation" (Kolb, 1984, p. 30). He also said "the combination of all four of the elementary learning forms produces the highest level of learning" (Kolb, 1984, p. 66).

This integration of the four modes of learning from his LSI formed the basis for Kolb's (1984) experiential learning theory (p. 140). As learners become more adept at using each of the learning styles, they begin to integrate these styles into a cohesive approach to learning. Kolb (1984) identified three stages of this process: acquisition of basic learning abilities, specialization in a specific learning process, and finally an integration phase that brings “a holistic developmental adaptive process … that is integrative in its structure” (p. 146). Kolb (1984) remarked that it is the role of higher education to teach learners to integrate different learning styles into a cohesive learning approach.

Kolb’s (1984) learning cycle has been criticized for being “polarized” (Mezirow, 1990, p. 6) and “too neat and perhaps over-simple” (Jarvis, 1987, p. 18). For example, Jarvis (1987) proposed a scenario in which a person was considering abstract mathematical concepts and reflecting upon them. Learning may occur in this situation without concrete experience or experimentation occurring. Jarvis (1987) also pointed to Schön’s (1983) own theory of reflection-in-action as an example of multiple steps of Kolb’s (1984) cycle happening simultaneously. However, when Kolb’s (1984) comments about the effective learner switching between different learning abilities are considered, it is evident that Kolb did not prescribe a rigid application of all four stages of his learning cycle. Kolb’s (1984) mention of an educational discipline’s tendency to lean towards certain combinations of learning styles also seems to confirm that he thought learning style specialization was inevitable (pp. 85-86).

Though he was critical of Kolb’s (1984) model as being too simplistic, Kolb’s research influenced Jarvis’ (1987) own model of adult learning considerably. Jarvis (1987) proposed an expanded view of learning as a state machine with no less than nine different states inside of it. Jarvis’ (1987) model, though complex, provides for many different approaches to learning. A person’s learning experience is influenced by its situation and context, and can consist of a combination of learning stages such as practice, evaluation, reflection, and memorization. The result of this experience is either a person who has been reinforced but unchanged or a person who has been changed and is now more experienced (Jarvis, 1987, pp. 24-35).

Boud et al. (1985) described a reflective process that is also similar to Kolb's (1984) in that it has distinct stages of experience, reflection, and outcome. Their model differed, however, in their attempt to further separate the act of reflection into its various components. It also emphasized the role that emotions play in the reflective process, which is something that was absent from many other learning models. The authors described reflection as an after-the-fact activity "in which people recapture their experience, think about it, mull it over and evaluate it" (Boud et al., 1985, p. 19). They suggested that reflection is a purposeful activity that can be instrumental in learning from experience and that reflection should be explicitly promoted in learning institutions.

Boud et al.'s (1985) process of reflection begins with the identification of an experience to be learned from. This experience should be revisited and replayed "in the mind's eye" (Boud et al., 1985, p. 27) to observe what happened. The learners should pay attention to the feelings that they had during this event, attempting to emphasize the good feelings while isolating the negative ones so that they can think objectively about the action. They should then re-evaluate the experience to see what can be learned from it and what actions can be changed.

Reflection and feedback. Each of these reflective theories and their cyclical patterns carry with them the concept of feedback. In every case there is an expectation that information from an action will reach the user in order for them to reflect on it. The concept of feedback is very important to the psychological approach of behaviorism. One of the earliest and most influential researchers to discuss feedback was Thorndike (1898/1998). He proposed the Law of Effect, which stated that connections between a cause and an effect could be reinforced or diminished based on the outcome of the action. This research was the inspiration for research in operant conditioning such as the work of Skinner (1935).

More recent research into feedback has shown that there is not always a clear link between improved feedback and student performance. In a study of the relationship between feedback interventions and performance, Kluger and DeNisi (1996) defined feedback as "actions taken by (an) external agent (s) to provide information regarding some aspect(s) of one's task performance" (p. 255). Their review of previous feedback interventions showed that negative effects were observed in over one-third of the studies. The studies they reviewed involved a participant's knowledge of the results of their effort, which is similar to the type of feedback mentioned in reflective theories. Though feedback interventions have remained popular, the authors mentioned that their variability could be due to many other factors, including the attention a student is giving to the task at hand.

There are also many types of feedback that an instructor can give a student. Wolsey (2008) attempted to classify different types of feedback that were given to students in an online writing class. He identified several different categories of written feedback that could be given to a student, including simple and complex affirmation, editorial corrections, questions, and personal comments. He observed that feedback has an interactive nature to it and was much more than just "identifying errors and expecting students to make corrections" (Wolsey, 2008, p. 313).

Wolsey (2008) also distinguished between formative and summative feedback, and said that students may have difficulty distinguishing between the two. Formative feedback is feedback that is available to students before a final grade has been determined. It is corrective and instructive without passing a final judgment, whereas summative feedback involves evaluating the work at hand. Formative feedback tends to encourage a reflective approach to learning. Students are given information about their learning experience and are allowed to correct and learn from their mistakes before proceeding to a final, graded work.

Reflective and non-reflective action. It is possible for an individual to perform a skilled action without thinking about it. For example, a person riding a bicycle is doing something subconsciously using a skill that took much practice to master, yet their mind is doing it without any conscious thought. Schön (1983) spoke of the phenomenon of "knowing-in-action" by skilled actors. He said, "although we sometimes think before acting, it is also true that in much of the spontaneous behavior of skillful practice we reveal a kind of knowing which does not stem from a prior intellectual operation" (Schön, 1983, p. 51).

Many researchers have made a distinction between doing an action without thinking about it and consciously thinking about the action while we do it. Langer (1989) described states of *mindlessness* and *mindfulness* that can each affect a person's actions. She listed several possible arguments for the causes of mindless activity such as repetition, context, and a belief in limited resources or time. Langer (1989) also described ways that people can be more mindful of their actions, such as changes in perspective, context, or a focus on the process of an action instead of its outcome. Mezirow (1991) referred to this concept as "reflection as mindfulness" (p. 114) and drew a direct connection between this and his own definition of reflective action.

In his transformation theory Mezirow (1991) also identified types of action, though he further classified non-reflective and reflective action into more specific types. Non-reflective action is broken down into *habitual action*, *thoughtful action*, and *introspection*. Habitual action is very similar to Langer's mindless action and can describe any action that we are not focused on. Thoughtful action "involves higher-order cognitive processes to guide us" (Mezirow, 1991, p. 106), though this sort of action can happen without reflection. A person may perform a skilled task and be fully conscious of their actions without reflecting on them. As Mezirow (1991) said, "cognition is not the same as reflection … we resort to reflection only when we require guidance in negotiating a step in a series of actions or run into difficulty in understanding a new experience" (p. 107). Introspection, which involves "thinking about ourselves, our thoughts or feelings" (Mezirow, 1991, p. 107), may also accompany thoughtful action but still does not imply reflection, only self-awareness.

Mezirow (1991) distinguished between different types of reflection that may occur, including reflection on content, processes, a premise, or a theory. The most basic type of reflection is that which is focused on the content of a problem or a process. It involves thinking about what we are doing or how we are doing it. This sort of reflection can "become an integral part of the process of thoughtful action…or it can occur only when the action stops because of a block, in which case it becomes part of a retrospective assessment [of our own processes]" (Mezirow, 1991, p. 107). Mezirow (1990) referred to reflection that occurred after the fact as *ex post facto reflection*.

An even higher level of reflection is reflection that questions assumptions that we have about a problem or our motivations to engage in a particular action. Mezirow (1991) called this *premise reflection* or *critical reflection*. This sort of reflection may "cause us to be critical of epistemic, social, or psychological presuppositions" (Mezirow, 1991, p. 108) and can be directly correlated to Dewey's connection between reflection and critical thinking (Dewey, 1910). The result of this type of critical reflection may be a *perspective transformation*, in which "reflection on one's own premises can lead to transformative learning" (Mezirow, 1990, p. 18).

Reflection-in-action. In his books *The Reflective Practitioner* and *Educating the Reflective Practitioner*, Schön (1983, 1987) proposed the existence of a type of reflective action that is distinct from ex post facto reflection or thoughtful action. He used the term *reflection-in-action* to describe the act of consciously thinking about an action while participating in it. Schön (1983) described reflection-in-action as “central to the ‘art’ by which practitioners sometimes deal well with situations of uncertainty, instability, uniqueness, and value conflict” (p. 50). In these works, Schön presented an argument for transitioning from valuing and teaching a purely scientific method to a mixture of science and artistic coaching that can better train students to deal with the uncertain issues that await them. The general application of reflection in a professional situation is known as *reflective practice* (Schön, 1987).

Schön (1987) compared the exceptional performance of everyday professionals to the work of artists and observed that this artistic talent could further enhance the successful application of scientific knowledge. This artistic talent might take several forms, including “an art of problem framing, an art of implementation, and an art of improvisation—all necessary to mediate the use in practice of applied science and technique” (Schön, 1987, p. 13). Schön (1983) used the example of artists such as baseball players or jazz musicians who are able to adjust their actions based on the feedback that they get during their performances. This unique ability to think about and learn from our actions while engaged in an activity is the basis for the concept of reflection-in-action.

Reflection-in-action happens when things don’t go as expected. Schön (1983) described it this way:

> Much reflection-in-action hinges on the experience of surprise. When intuitive, spontaneous performance yields nothing more than the results expected for it, we tend not to think about it. But when intuitive performance leads to surprises, pleasing or unwanted, we may respond by reflecting-in-action. (p. 56)

Once the practitioner notices that something isn't going according to plan, he or she will respond by conducting an on-the-spot experiment of some kind. The experiments that can be conducted in-action, however, aren't of the kind that is found in scientific journals. The practitioner is not constrained by the formal rules of research, but makes quick observations about what has changed and what may be causing that change. This cycle of experimentation is similar to the integration of learning described in Kolb's experiential learning theory of development (Kolb, 1984).

Several examples of reflection-in-action and reflective practice were given throughout Schön's works. Schön (1983) generalized each of these examples into a common pattern of inquiry. When practitioners first encounters a problem, they "impose a frame on it" (Schön, 1983, p. 269), which means to try to look at it from a particular point of view. The practitioners follow the results of this imposed frame while remaining "open to the situation's back-talk" (Schön, 1983, p. 269). Whenever the practitioners are surprised by the results of the situation, they alter their existing frame with "new questions and new ends in view" (Schön, 1983, p. 269). The practitioners are often able to make connections between a unique situation and a similar situation that he has experienced before, and then are able to form a new series of experiments to test this possible connection.

Educating for reflective practice. Schön (1987) suggested the educational concept of the practicum as a place where reflective practice can be taught. He identified three different kinds of practicums. Some, such as those used in computer science and chemistry, are where "facts, rules, and procedures [are] applied nonproblematically to instrumental problems" (Schön, 1987, p. 39) and the practicum is a type of technical training. Another type of practicum that is popular in the study of law and medicine is where students are subjected to verbal drills and case studies designed to train them to think like a practitioner in their field. The third type of practicum can often be found in schools of art and design. These practicums "depend for their effectiveness on a reciprocally reflective dialogue of coach and student" (Schön, 1987, p. 40). It is on this third example that Schön focused most of his discussion regarding educating for reflective thought.

An example that Schön (1987) used to describe this sort of reflective practicum is one of an architecture student learning from a teacher. The student is faced with a problem that she has begun to have difficulty with. The instructor then takes the sketchpad and begins to help her re-frame her problem. Schön (1987) used the artist's sketchpad as a metaphor for reflection-in-action, because "here they can draw and talk their moves in special action language…. Because the drawing reveals qualities and relations unimagined beforehand, moves can function as experiments" (p. 77). This virtual world allows the student and teacher to easily try new approaches, examine the results, and continue to make new experiments until a suitable design has been reached.

In Schön's (1987) architecture example, the students interviewed expressed a frustration with the simultaneous process of learning about design and participating in design classes. The student came to the realization that "she is expected to learn, by doing, both what designing is and how to do it…. And although [others] may help her, *she* is the essential self-educator" (Schön, 1987, pp. 83-84). It could be said that these students don't know what they don't know, but the act of reflective practicums allow them to engage with the material. With the help of a coach they are slowly able to construct their own repository of professional knowledge and experience.

Applications of reflective practice. The idea of reflective practice has had a great impact on several fields of research since its introduction. This is true particularly in "professions such as nursing, social work, planning, psychology and psychotherapy, which have long grappled with aspects of their practice that could not be easily reduced to fixed and testable scientific theory" (Redmond, 2006, p. 31). Several studies in the fields of nursing and health education have shown a special interest in integrating the concepts of this theory into their curriculum (Mann, Gordon, & MacLeod, 2009). A large number of researchers in teacher education have also been influenced by the idea of reflective practice as well (Hatton & Smith, 1995).

Hatton and Smith (1995) studied the application of reflective learning concepts in teacher education. They found a wide variety of approaches, but few attempts to actually measure the presence of reflection. The researchers conducted several studies, interviewing students from each academic year and then analyzing and coding student essays for evidence of reflection. They synthesized the findings of several of these previous studies into their own operational framework of reflective thought and identified three main categories of reflection.

The first and lowest category identified by Hatton and Smith (1995) was focused on *technical rationality*, often the result of feedback from a training situation. This sort of reflection only addressed how an assignment was performed, usually clouded by the student's "personal worries" (Hatton & Smith, 1995, p. 45) about their performance. The next category of reflection was *reflection-on-action*. This reflection could take a variety of forms, but each variation was performed after the action has taken place. *Reflection-in-action* was described by Hatton and Smith (1995) as "the most demanding type of reflecting upon one's own practice" (p. 46). Any or all of the previous types of reflection could be incorporated into reflective thought that happens while an act is being performed. This action was described as a practitioner's ability to consciously "think about an action as it is taking place, making sense of what is happening and shaping successive practical steps using multiple viewpoints as appropriate" (Hatton & Smith, 1995, p. 46).

Most of the approaches at encouraging reflective learning attempted to integrate reflective practice concepts directly into the instructional process. Some of these approaches, such as Redmond's (2006) Reflective Teaching Model, encouraged a complex form of emancipatory learning where the student placed themselves in the position of customers interacting with a service provider. Others simply asked students to write their thoughts about experiences in journals, like Hatton and Smith (1995), or asked them to teach small lectures and then discuss their experiences with peers and their instructors (Cutler, Cook, & Young, 1989).

Measurement of Reflective Thought. Kember et al. (2000) noticed the proliferation of attempts to encourage reflective thought in education and observed "how little attention has been paid to methods for assessing whether students do engage in reflective thinking and if so to what extent" (p. 382). The researchers adapted Mezirow's (1991) classifications of action and reflection into categories that could be more easily distinguished from one another. This resulted in an instrument that measured four distinct dimensions of a students' level of reflective thought: habitual action, understanding, reflection, and critical reflection. Kember et al.'s (2000) dimension of habitual action remained virtually unchanged from Mezirow's (1991) definition and was described as any action that the person is not actively focusing on. Since thoughtful action represented a very broad range of cognitive activity (including reflective action), the authors chose to restrict this dimension to Bloom's (1984) definition of comprehension, which they chose to rename *understanding*. Bloom (1984) defines this as the lowest level of understanding where the learner could “make use of the material or idea being communicated without necessarily relating it to other material” (p. 204).

Kember et al. (2000) retained Mezirow’s (1991) distinction between simple reflection and critical reflection. Mezirow’s (1991) definitions of content reflection and process reflection were grouped into one single dimension called reflection. This concept of reflection was reinforced by definitions obtained from Dewey (1910) and Boud et al. (1985). The more complex dimension of critical reflection was derived from Mezirow’s (1991) “premise reflection” which he has elsewhere referred to as critical reflection (Mezirow, 1990).

Once the instrument had been created and pilot tested, Kember et al. (2000) administered a final version to 303 students from Hong Kong studying in healthcare related fields. The results of this study indicated some significant relationships between the four dimensions being studied, though these relationships were expected. For example, a significant correlation was observed between habitual action and critical reflection. The authors attributed this to the type of professional practice described by Schön (1983) in which a practitioner would act habitually until a confounding problem arose, causing them to critically reflect on the situation. Students who engaged in either type of reflection were also more likely to study for understanding, “particularly in more theoretical parts of a course, which have less obvious relationships to practice” (Kember et al., 2000, p. 389).

Reflective thought has had a large impact on many fields of education, but the field of interest in this study is computer science education. Edwards (2004) has shown that automated software testing along with TDD has had a positive impact on introductory computer science students and has hypothesized that this is due to an increased level of reflection-in-action on the part of the student. The next section of this chapter will explore the history of research in automated testing and TDD. It will also examine Edwards' (2004) theory linking reflection to increased academic performance when using TDD.

Automated Grading in Computer Science Education

Automated grading in computer science has been a goal since computer science was first taught in an educational setting. The rationale is simple: computers are good at doing tedious things over and over again, so why not use a computer to grade student assignments? In practice this has not turned out to be as easy as it appears, though a certain level of success and automation has been achieved over time. A review of automated graders conducted by Douce et al. (2005) identified three major generations of automated grading systems: early assessment systems, tool-oriented systems, and web-oriented systems. The authors of this review also identified the emergence of a more recent group of systems called *meta-testers*. Many such systems have been developed over the years; examples from each of these categories of systems will be presented.

Early assessment systems. The first automated tools were oriented towards the act of grading assignments. These tools functioned as batch-processed jobs, similar to other early computer programs. Student work was fed through the grading system in batches and the computer would provide the resulting analysis. The earliest example of an automated grader was Hollingsworth's (1960) assembly language grader; this system functioned using punch cards, and would only return a message stating "WRONG ANSWER" (p. 528) along with some basic indication of the source of the problem or "PROBLEM COMPLETE" (p. 528) if the program executed successfully. Students were allowed to submit a stack of punch cards on a daily basis to be run in a batch, and their results were returned the following day. In what may be the earliest mention of distance education in computer science, Hollingsworth (1960) mentioned: "We currently have a student who is doing the exercises by mail. He sends his programs and corrections, we send him grader results. It is still too early to know how well this procedure works" (p. 529).

A more sophisticated example of this type of early grader was Aaronson's (1973) Automated Grading System for the Instruction of COBOL Programming (AGSICP) system, which was developed for the automated grading of COBOL code. This system required an instructor to provide a reference program and then it would run a series of diagnostics against the reference program and each student's program. The system was able to assign partial credit and let the instructor know which lines of code had errors. It does not appear that this system was available to students, but rather was used by graders to quickly assess student code submissions.

Tool-Oriented Systems. Douce et al. (2005) defined tool-oriented systems as "pre-existing tool sets and utilities supplied with the operating system or programming environment" that were offered to the students as either command-line or graphical user interface (GUI) programming tools (p. 3). These sorts of tools were made possible by the interactive computing environments that became available in the 80s and 90s, and they allowed students to run tests on their own computer programs in real time.

A good example of this sort of interactive testing tool was the "TRY" system developed by Reek (1989) at the Rochester Institute of Technology. This system was one of the first such interactive systems available to students, and was an accessible program on the university's computer system. The tool had access to a collection of instructor-provided test cases but prevented the students from seeing all the test cases. The results of the tests were written to a log file in the instructor's account, and the instructor could choose to provide the students with a limited view of the results as well. Jackson and Usher's (1997) Assessment System (ASSYST) program was an even more sophisticated testing tool. Not only did it use test cases to evaluate the correctness of student code, it also had mechanisms for evaluating efficiency, complexity, style, and code coverage of student-supplied test cases. The ASSYST program also had a graphical interface that allowed for point-and-click interaction with student code.

Web-Oriented Systems. Web-oriented systems improved on the features present in tool-oriented systems, but their real innovation was the ability for students to submit their code to automated testers through the Internet wherever they happen to be working on their assignments. Online systems such as the BOSS Online Submission System (BOSS; Joy, Griffiths, & Boyatt, 2005) and CourseMarker (Higgins, Hegazy, Symeonidis, & Tsintsifas, 2003) are good examples of modern client-server web applications designed to allow students to submit their code to an online system and have it evaluated for correctness and style using similar techniques as the ASSYST program. These systems provide online grading tools that allow instructors to see who has submitted assignments and statistics about their class' efforts.

Meta-Testers. More recently, educators have developed web-based automatic testers that have the ability to test the students code as well as student-supplied test cases. This new class of tools is known as meta-testers because they can test the students' code as well as the student's tests. With a meta-tester, a student's grade can be based not only on their program's correctness but also on the completeness of their own test cases.

Edwards' (2003) review of existing automated testing systems concluded that students were not encouraged for performing testing on their own and that they relied too heavily on the instructor-provided sample data and test cases. He said that students needed "explicit, continually reinforced practice in hypothesizing about the behavior of their programs and then experimentally verifying (or invalidating) their hypotheses" (Edwards, 2003, p. 148). To facilitate this sort of thinking, Edwards (2003) advocated a test-first approach where students would create their own test cases and submit them for grading alongside the instructor's test cases. By grading the code this way, this system would place the burden of proof for correctness on the student's own tests.

Edwards (2003) developed a web-based grading system called Web-CAT that evaluated software on correctness, test completeness, test validity, and code quality by using a variety of commercial software evaluation tools. Both students and instructors used the JUnit testing suite to write test cases to evaluate the code. Code completeness was determined by running the student test cases, though instructor supplied set of tests could also be used to determine if the students tested the code thoroughly. Validity was determined by "running the student tests against an instructor-provided reference implementation" (Edwards, 2003, p. 148). Other tools were used to evaluate the level of existing code that is actually executed and the formatting and documentation of the code itself. Each of these dimensions was combined into a final score that was presented to the student, though a portion of the grade could be reserved for a human grader to evaluate later. Students were free to use the system as often as they liked to evaluate their code and then make corrections based on the testing results.

Edwards tested this software on two junior level programming courses at Virginia Tech (Edwards, 2004). The control group for his experiment was an earlier class that had submitted their assignments online through an electronic grading system that did not provide detailed feedback but simply tested against a set of test cases and provided a score. The experimental group of students used the new Web-CAT tool and wrote their own test cases to be used for grading. The experimental group was shown to have significantly higher project scores, significantly fewer test case failures, and to have started coding their software an average of two days before the control group.

The Marmoset grading system (Spacco, 2006) was another web-based grader that was built on Edwards' research. Like Web-CAT, Marmoset also supported instructor and student provided test cases but it also introduced the concept of release testing. Release testing was intended to simulate the feedback that can be obtained by taking prototypes to a client for testing. This was achieved by giving the students limited access to a set of additional test cases once they successfully passed all the initial public test cases provided by the instructor. Seventy students were surveyed in a study conducted by Spacco et al. (2006) to ascertain the effectiveness of the Marmoset system. The students' responses indicated that they overwhelmingly preferred "release testing vs. post-deadline [testing]" and that they were "encouraged to start work early" by the introduction of release testing (Spacco et al., 2006).

In a later paper, Spacco and Pugh (2006) observed that their students were creating test cases after they had completed writing their code. This was not in accordance with his TDD approach, so the authors made further modifications to the Marmoset system that included removing the names of the release tests and also adding a dynamic feature to the system that released more test cases as the student provided more of their own.

Software Testing

With the increased emphasis on automated testing in academic computer science instruction has come a similar interest in automated testing in the area of commercial software development. Testing has always had an important role in software engineering, though it has sometimes received less than wholehearted participation from programmers (Beck, 2003). A recent movement in software design called TDD (Beck, 2001) has evolved into an effort to place testing at the forefront of the software development process.

Test-driven development. The process of TDD has its origins in the extreme programming movement. It can be described by its two rules: “Write a failing automated test before you write any code…[and then] remove duplication” (Beck, 2003, p. xix). Software developers who use this style of coding are first encouraged to write test cases for their program before beginning to implement it. Once a failed test case is written, the programmer is then permitted to write just enough code to get the test to pass. Once that has been achieved, the programmer then optimizes the code to remove any inefficiency that might have been introduced while trying to make the code functional. Beck claimed that TDD reduces programmer fear, which can result in increased communication, encourage programmers to seek helpful feedback, and inspire decisive action in the face of difficult situations (Beck, 2003).

Test-driven development is really more about design than testing. Janzen and Saiedian (2005) emphasized the impact that TDD has had on software design, as well as the dramatic shift that is required on the part of the programmer to use this approach. They observed that “program testing has traditionally assumed the existence of a program” (Janzen & Saiedian, 2005, p. 44), and the idea of using tests to decide how to design a program was a “radical concept” (p. 44) for many programmers. Beck (2003) has used several metaphors to explain the impact of testing on software development. In his analogy of pulling up a heavy bucket of water from a well he calls testing “a ratchet mechanism to enable you to rest” while cranking (Beck, 2003, p. xi). He also called testing “a canary in a coal mine” (Beck, 2003, p. 194), alerting the programmer to potential problems caused by changes made elsewhere in the system.

Effectiveness of TDD. Quite a bit of work has been done to integrate the concepts of TDD into computer science education, though the evidence of its effectiveness is still inconclusive. Many previous research studies have examined the influence of TDD on external code quality, internal code quality, and productivity, while very few have focused on TDD's influence on student performance and learning (Kollanus, 2010). Reviewers of the literature in this area have highlighted the lack of properly controlled experiments with sufficient numbers of participants (Janzen & Saiedian, 2005; Kollanus, 2010). Instructors have reported that the concept of TDD was hard to grasp for many students (Keefe, Sheard, & Dick, 2006). Several of the experiments in this area found no significant evidence of improvement in code quality or productivity using TDD (Huang & Holcombe, 2008; Kollanus, 2010; Muller & Hagner, 2002).

The process of introducing TDD in curriculum is also a point of contention in the computer science community (Desai, Janzen, & Savage, 2008). Some instructors have argued for the integration of TDD into the curriculum from the very beginning of an introductory class (Christensen, 2003; Edwards, 2003). Another researcher found more success in introducing the topic of testing gradually throughout the semester and remarked that students wrote more test cases "later in the semester after they had seen a number of examples and had feedback on their efforts" (Leska, 2003, p. 168). Keefe et al. (2006) also recommended a more traditional approach to testing earlier in the course, followed by an introduction of TDD later in the semester. Desai et al. (2008) called this an "incremental instructional approach" (p. 100).

Any perceived improvements in quality or productivity may possibly be explained by increased effort or skill on the result of the participants. Erdogmus, Maurizo, and Torchiano (2005) conducted an experiment examining test-first vs. test-last software development. The authors did not discover improvements in software quality, though the students did tend to write more tests and therefore more code, which would account for the measured increase in productivity. Students who wrote more test cases also tended to spend considerably more time testing their application (Huang & Holcombe, 2008). Barriocanal, Urban, Cuevas, and Perez (2002) made the use of test cases optional in their experiment and observed that only about 10% of participants chose to do so. The few students who did write test cases had previous experience and as a result scored better on the projects.

Despite the poor results observed in many of the TDD studies, several researchers noted that other benefits might occur when using these methods. Muller and Hagner (2002) did not observe any improvements in code reliability or coder productivity, but they did notice a tendency for TDD developers to re-use more of the code they had written. This may be because they already had test cases developed for these methods. Marrero and Settle (2005) also did not observe any significant increase in student grade performance between those who wrote test cases and those who did not. They did observe some qualitative benefits, however. The students who were asked to write test cases were forced to think more about interacting with their code which helped them design the software while thinking about the tests before they began coding it. This kind of thinking may be associated with an increase in reflective thought associated with good testing and design.

Reflection in Computer Science Education

Edwards (2004) has made a connection between the concept of reflection and the kind of thinking that happens when a computer science student creates a software test and runs it against a piece of code. He identified the prerequisites for this sort of thinking as the ability to "predict how changes in code will result in changes in behavior…[and also] continually reinforced practice hypothesizing about the behavior of their programs and then experimentally verifying (or invalidating) their hypotheses" (Edwards 2004, p. 24). Edwards (2004) then listed five perceived roadblocks that keep computer science educators from adopting software testing in their classes, though he does not completely address all of them. These roadblocks were a student's need to master other skills before learning testing, instructor reluctance to learn and teach a new topic, inability to grade student test cases, inability to provide rapid feedback, and a student's need to see value in any new topic that is introduced.

Edwards (2004) offered up automated grading software as an answer to the problems of instructors being too busy to grade test cases and a student's need for "frequent, concrete feedback" (p. 27). Because students can submit their own test cases to an always-available testing engine, they do not burden the instructor with extra grading work. A suite of instructor-provided test cases would allow the student test cases to be compared to a reference suite to test for completeness and accuracy. The availability of this instantaneous feedback was also Edwards' (2004) answer to the question of student value, since the students would be able to get feedback on the success of their code while they were developing it instead of waiting for days after the code has been submitted.

Edwards (2004) points to the ease and approachability of the JUnit testing software as an answer to the concern of testing being yet another item competing for student attention and classroom hours. In the experiments that he conducted, the only extra time devoted to JUnit instruction was "one lecture hour of course time and several reading assignments outside of class" (Edwards, 2004, p. 28). His students seemed to adapt to the JUnit tool itself easily.

The first roadblock identified by Edwards (2004) was left unaddressed, however. Students who are just learning to program are wrestling with many introductory concepts, and the idea of creating a test before writing the code may be foreign to them. This difficulty was also observed by Marrero and Settle (2005) and was the reason behind many other instructors recommending a delayed introduction of TDD concepts (Keefe et al., 2006; Leska, 2003). Edwards (2004), however, argued that adopting these methods in an introductory course would require little extra work on the part of the instructor, yet provide the opportunity for great benefit.

The work started by Edwards (2004) is important and insightful, but it falls short in several areas. Edwards (2004) was unable to tell if using an automated grader alone made a difference in student work because both of his groups were using automated graders of some variety. Edwards (2004) also did not attempt to verify his theory that reflective thought was the reason that the performance of his students had increased.

Chapter Summary

The concept of reflection is an important one in educational research. Researchers such as Dewey (1910), Kolb (1984) and Mezirow (1990) have all reserved an important place in their theoretical constructs for a student's reflection upon new material. Schön (1983) introduced the idea of reflection-in-action, which is the act of thinking about an action or a process while still engaged in it. Edwards (2004) connected the idea of reflection-in-action to automated software testing in computer science, and suggested that it was a potential reason for improved student performance. Automated software testing in academic settings has evolved to now include TDD-inspired activities that provide immediate feedback to the students. More research is needed to determine if student-provided test cases are necessary to see a benefit in student performance, and whether or not this feedback will result in increased levels of reflection.

CHAPTER III

METHOD

This research study used a self-selecting, between-subjects design. Groups of students in existing introductory computer programming classes were studied. Student data were collected through paper surveys and grade data were collected electronically from instructors. These data were examined to look for relationships between automated software testing, levels of student reflection, and student performance on programming assignments.

The independent variable in this study was the introduction of an automated software-testing system for a portion of each semester. The first dependent variable was the student's performance during each of the three programming assignments. Another series of dependent variables was student responses to a survey consisting of four scales that measure a student's level of reflective thought: habitual action, understanding, reflection, and critical reflection. These four variables were grouped according to usage of the experimental system and then examined as potential mediator variables for student performance on the assignments. There were five possible moderating variables: gender, age, major, student classification, and whether it was their first computer science class. Each of these variables was analyzed to look for unintended effects on student grade performance or reflection.

Research Questions and Hypotheses

Six research questions were used in this study: three questions to investigate the interactions between the independent and dependent variables and three to investigate the possible influence of the moderating variables on the dependent variables. Each of these questions was further broken down into hypotheses to aid in statistical testing. This was especially important for questions involving reflection where four separate dependent variables each needed to be tested for statistical significance.

Research Question 1. How does the use of automated software testing influence levels of reflective thought in students compared to students who do not use automated software testing?

H_1. Average self-reported levels of habitual action will be significantly different for students who use an automated software-testing environment compared to those who do not.

H_2. Average self-reported levels of understanding will be significantly different for students who use an automated software-testing environment compared to those who do not.

H_3. Average self-reported levels of reflection will be significantly different for students who use an automated software-testing environment compared to those who do not.

H_4. Average self-reported levels of critical reflection will be significantly different for students who use an automated software-testing environment compared to those who do not.

Research Question 2. How does the use of automated software testing influence student performance on introductory computer science programming compared to students who do not use automated software testing?

H_5. Average student performance on programming assignments will be higher for those who use an automated software-testing environment compared to those who do not.

Research Question 3. To what degree does reflective thought affect student performance on programming assignments for those who use automated software testing compared to those who do not?

H_6. There will be a significant relationship between self-reported levels of habitual action and average student performance for students who use automated software testing compared to those who do not.

H_7. There will be a significant relationship between self-reported levels of understanding and average student performance on programming assignments for students who use automated software testing compared to those who do not.

H_8. There will be a significant relationship between self-reported levels of reflection and average student performance on programming assignments for students who use automated software testing compared to those who do not.

H_9. There will be a significant relationship between self-reported levels of critical reflection and average student performance on programming assignments for students who use automated software testing compared to those who do not.

Research Question 4. How does demographic data influence levels of reflective thought in students both who use and who do not use automated software testing?

H_{10}. A significant relationship does not exist among age, major, classification, gender and self-reported levels of habitual action**.**

H_{11}. A significant relationship does not exist among age, major, classification, gender and self-reported levels of understanding**.**

H_{12}. A significant relationship does not exist among age, major, classification, gender and self-reported levels of reflection**.**

H_{13}. A significant relationship does not exist among age, major, classification, gender and self-reported levels of critical reflection**.**

Research Question 5. How does demographic data influence student performance on introductory computer science programming assignments?

H_{14}. A significant relationship does not exist among age, major, classification, gender and performance of students on introductory computer science programming assignments.

Research Question 6. How does demographic data influence student usage of an automated software-testing environment?

H_{15}. A significant relationship does not exist among age, major, classification, gender and student usage of an automated software-testing environment.

Research Setting

The setting for this study was The University of West Florida: a mid-sized public university in the southeastern United States. The university has an enrollment of 12,588 students. Student classification includes 10,158 undergraduate students and 2,430 graduate and doctoral students. The population of students is diverse and approximately 68.03% of students identify themselves as Caucasian, 12.42% as African American, 8.18% as Hispanic, 3.03% as Asian, 3.5% as international or unreported, 0.69% as American Indian or Alaskan, 0.39% as Hawaiian or Pacific Islander, and 3.77% from two or more ethnicities (University of West Florida, 2014).

Participants

The participants for this study were selected from undergraduate students in introductory Java programming classes at the university during the Spring, Summer, and Fall 2013 semesters. These classes were offered using a traditional classroom setting, and the students had several programming assignments to complete throughout the course of the semester. Students in these classes came from a variety of backgrounds and majors and had varying levels of previous experience in computer programming. The instructors for these classes participated by allowing this study to take place within their classrooms. They also participated in the design of the projects to make them suitable for the automated grader to evaluate the assignments.

Permission was obtained from several entities in order to conduct this research. The Institutional Review Board (IRB) evaluated and authorized this study (Appendix A). The instructors teaching the classes agreed to modify their projects and collect data for analysis. Students were asked to participate in the study, and their participation was completely voluntary. Those choosing to participate were asked to sign a consent form that informed them of the details of the study (Appendix B). Students were informed that their identity would remain confidential, and that those who declined to participate in the study would not be negatively impacted in the class in any way. They would simply complete the class using traditional methods.

Instrumentation

Data were collected from the participants in a variety of ways. Survey data were collected using paper forms and entered into a statistical program for analysis. Student assignment submissions were stored in a secure database belonging to the automated Web-CAT grading software. Instructors compiled student performance data and submitted these data to the researcher in a digital format for further processing. These data were converted and stored in a format readable by the statistical software.

Demographic survey. Students were asked to complete a short demographic survey designed to gather their university email address, gender, age, major, and classification (Appendix C). There were eight questions on this survey. The students were also asked if they would like to receive the results of the study, and their response to this question was recorded as well. The purpose of this survey was to collect information about the type of students enrolled in each class in order to analyze the effects that these demographics might have on other dependent variables like reflection and student performance.

Reflective thinking survey. Kember et al. (2000) have created an instrument designed to measure levels of reflective thinking in university students (Appendix D). Their questionnaire measures student's levels of thinking on four scales: habitual action, understanding, reflection, and critical reflection. There are four questions in each scale, for a total of 16 questions in all. Although it was developed for use with students in healthcare-related majors, the questions are phrased in such a way that they could apply to any university-level course. The authors gave permission to use their survey in future academic work, provided that they were properly credited with originating the survey.

Reliability and validity. Several steps were taken by Kember et al. (2000) to examine the reliability and effectiveness of their instrument. Preliminary versions of the survey were tested in order to fine-tune the questions on the survey. Over three hundred health sciences students at a major university in Hong Kong completed the final version of this instrument. The results of this study were then analyzed to examine its reliability and validity.

A Cronbach's alpha analysis was performed on the data from the authors' final study to determine if each of the four scales could be considered internally reliable. The values for each of the four scales were: Habitual Action, 0.621; Understanding, 0.757; Reflection, 0.631; and Critical Reflection, 0.675. Some of these values may be considered questionable because they fall below the traditional threshold of 0.70 (Nunnally & Bernstein, 1994), but the authors endorsed the scales and said that they had acceptable levels of internal reliability. The low number of items for each scale may also contribute to the slightly lower than expected Cronbach's alpha values. In a later study, Leung and Kember (2003) used this questionnaire and achieved similar alpha values. The authors cited work done by Schmitt (1996) that further confirmed the acceptability of lower alpha values, especially in cases such as this where multiple dimensions are being used to measure one idea.

Confirmatory factor analysis was performed on the study to determine its model validity. The chi-squared test and Bentler's comparative fit index (CFI) model were used to compare the four factors of the model to a hypothesized model. The results were a Chi-squared value of 179.3 with 100 degrees of freedom and a CFI value of 0.903. These values confirm that this instrument passes these tests for model validity.

Automated software-testing tool. The Web-CAT automated software-testing tool was selected for this study. This selection was based on its ease of use and adoption by several prominent computer science departments in the United States. Amazon Web Services were used to set up a secure database and web server for this tool. Participants who used this resource were given access and training on how to use this tool. Data were collected on the number of submissions from each student and was used to group the students by level of participation. The outcome of each submission was collected as well, but was not used in the analysis.

Procedures

Four instructors teaching seven different sections of an introductory Java programming course agreed to participate in this study. Permission was obtained from these instructors to recruit students from within their classes. The researcher met with each section to introduce the study to the students. The students were informed that their participation was not required and that no penalty would be assessed for not participating.

Participant recruitment and assignment. For two sections of the Fall 2013 semester, an incentive was offered of 10 points for each project that the students participated in. The description of the study on the consent form was read to the class, and then students were asked to sign a consent form (Appendix B) if they wished to participate in the study. Only half of the students in the Spring 2013 semester were invited to use the Web-CAT software. When a very low number of students elected to use the Web-CAT software that semester, use of the Web-CAT tool was opened up to all participants in the Summer and Fall 2013 semesters.

Demographic survey. All students who consented to participate in the study were given a demographic survey (Appendix C) in the same class period that the consent form was collected. The survey was one page long and was completed quickly by the students. Once both the informed consent form and demographic survey were completed, the students were instructed to continue in the class as normal until they were contacted later in the semester with further instructions on how to use the Web-CAT software.

Setting up Web-CAT. The Web-CAT automated software-testing tool was installed on a server hosted by Amazon Web Services and set up for use by each of the classes. Separate class sections were configured in Web-CAT for each of the sections being taught. Assignment collectors were set up for each section to collect the submissions from each student. During each semester, the instructors collaborated with each other and with the researcher to design project assignments that would work well with the testing software. Test cases were then written and sent to the instructors for approval. These test cases were then uploaded to the assignment collector to use for automated software testing. The students from each class were sent an email containing logins to the system based on their university email address and a randomly generated password. They were then given a training session on how to use these accounts to submit their work to the grader from within their software development application.

Instruction. All students in each section were taught using the same curriculum and basic project structure. Their instructors taught them the importance of iterative development, showed them software-testing strategies, and encouraged them to test their work frequently while completing their assignments.

The students given access to Web-CAT were contacted via email with information about the testing system. A training video and handouts on how to set up and use the software were emailed to these students. In order to encourage participation, the same training was also performed in class for students in the Summer 2013 and Fall 2013 semesters. These sessions were completed prior to the programming assignments being studied so that everyone was ready to use the testing system once they began.

This training session was scripted (Appendix E) to ensure a consistent presentation of the training material. The session included a demonstration on how to submit a project and notes on how to interpret the results of the automated software testing process. Students were given a username and password to log into the grader and were also shown how to configure their coding environment to connect to the grader and submit their assignments for evaluation (Appendix F). Students were encouraged to bring their laptops so that the configuration could be tested before they left. They were able to email the researcher to request additional assistance in configuring and using this software, and several were assisted individually in getting started with Web-CAT.

Curriculum and assignments. The instructors in each section of the course used the same textbooks, slides, and programming assignments. The students in all sections of this course were first given a chance to acclimate to the demands of the course and become familiar with the programming language. Programming assignments three, four, and five were then used for analysis in this study. For those given access to Web-CAT, additional instruction in the use of the Web-CAT tool (Appendix E) was given prior to these three experimental assignments.

For each of the three experimental assignments, students who had access to the automated testing tool were able to submit their code to the tool as often as they wished. Each submission was compiled and run against the test suite created by the researcher. Students were given immediate feedback based on the number of test cases that their software passes. For each test case that failed, students were given the name of the test case, which served as a hint for what they needed to improve on. Test cases were given names such as "testSumOfSquares" to indicate a method or feature being evaluated. Student code was also run against the Sun Coding Conventions for Java using the Checkstyle program. This program gave students feedback about the formatting of the code, including alignment, formatting, documentation, and code use.

All participants in this experiment submitted their assignments through the drop box feature of the university's online learning management system. This consistent method for submission ensured that the graders assigned to the classes would evaluate the submissions in an equal way. They were not able to differentiate between those who were participating in the experiment and those who were not.

Reflective thinking survey. After all assignment data were collected, each of the participants was given a survey of reflective thought (Kember et al., 2000) to measure their levels of self-reported reflection. Each question used a five-level Likert scale value. The original researchers included these instructions for scoring the instrument: "A student's score on each scale is computed simply by adding the response score for each of the four items. Strongly agree was scored as 5, through to strongly disagree as 1. Hence, the scores for the four scales could range from 4 (strongly disagree) to 20 (strongly agree)" (Kember et al., 2000). Student performance was computed by using the corrected (without bonus points) score for each of the specific projects.

Data storage. Once the informed consent forms were gathered, a crosswalk sheet of email addresses and identification numbers was generated containing an entry for each participant. Survey and grade data were initially collected using the participant's university-assigned email address as the primary means of identification. This information was translated into these assigned numbers to render an anonymous data set before anyone else saw it. When the data were gathered and saved in a finished state for analysis, only these assigned ID numbers were used. This process helped to protect the identity of the participants.

Instructors submitted the participants' grade performance for each of the assignments used in this study. The original survey data were collected on paper forms and stored in a secured location. Student programming assignment submission data were stored in an online database associated with the Web-CAT automated software-testing system. After the experiment was finished, the database was taken offline and the student submission data were saved in a backup image of the database server.

Once the data collection period was over, the data from each of these various sources was entered along with their assigned identification numbers, then exported to appropriate data files to be processed by statistical software. These files were saved on secure file storage provided by the university. These data were available only to the researcher and will be retained along with the data collected from the Web-CAT system for a two-year period following the study. When the time comes to destroy the data, the digital files will be securely wiped and no copies will be retained. All physical documents will be destroyed using a crosscut paper shredder.

Statistical Analysis

The participants in this study were separated into groups based on the usage of the automated testing tool as an independent variable. The data from each of these groups were analyzed using the Statistical Program for Social Sciences (SPSS) software to determine statistical significance that would allow confirmation or rejection of the previously stated hypotheses.

To answer the first research question regarding the influence of automated testing on a student's level of reflection, test scores were computed for each of the four different scales of the reflective thought instrument. A one-way multivariate analysis of variance (MANOVA) statistical test was performed for the hypothesis corresponding with each scale. In this test, the use of automated software-testing software was the categorical independent variable and self-reported levels of habitual action, understanding, reflection and critical reflection were the four continuous dependent variables. A one-way analysis of variance (ANOVA) test was used to examine the second research question regarding the influence of automated software testing on student performance. In this test the use of automated software-testing software was the categorical independent variable and average student performance was the continuous dependent variable.

For the third research question, results from the reflective thought instrument were separated into three groups based on their usage of the Web-CAT software. A Pearson's product-moment correlation was run for each group to determine if there was any correlation between reflective thought and student performance for each usage level. A factorial MANOVA was used for the fourth, fifth and sixth research questions to look for significant relationships between any of the demographic variables and reflection, student performance, or Web-CAT usage. Post-hoc analysis was also conducted where appropriate.

Chapter Summary

The goal of this research was to investigate the effects of automated testing software on levels of student reflection and student performance. This was a self-selecting, between subjects design that examined the performance of students in introductory computer programming courses at the University of West Florida. Those who participated were given the option of using Web-CAT software to evaluate the computer code that they wrote during the semester. Student reflection was then measured with the four dimensions of Kember et al.'s (2000) reflective thinking survey: habitual action, understanding, reflection, and critical reflection. Participants were then compared based on Web-CAT usage level, levels of reflective thought, and grade performance on programming assignments.

CHAPTER IV

RESULTS

The goal of this study was to examine the effect of automated software testing on student performance in an introductory computer science setting. Levels of reflective thought were also measured and examined for a relationship between reflection and student performance. This chapter contains the results of the study and is organized into two sections. The first section is a summary of the methods used to collect the data and a description of the participants who were involved in the study. The second section contains an analysis of the data that was collected to answer each of the research questions in this study.

Methodology Summary

This study involved the participation of students in seven introductory Java programming courses over the course of three semesters. All participants were given a basic demographic survey at the beginning of the semester. The students were given access to the Web-CAT automated software-testing tool and asked to submit projects 3, 4, and 5 to the tool in order to gain feedback on ways that they might improve their code. They were allowed to submit each project as often as they liked, and usage statistics were gathered during that time. At the end of the semester, a survey of reflective thought was administered to the participants in order to measure the level of reflection that they associated with the course. The participant's grades for each of these projects were also collected from their instructors in order to determine their performance in the course.

Participants and Demographics

Over the course of three semesters, 144 students volunteered to participate in this study. The results of the demographic survey were compiled and are summarized in Table 1. A large majority of the participants were male (79.2%), while only 20.8% were female. Most of the participants (81.3%) reported their classification as Sophomore or Junior, while the split between computer science majors and non-majors was more even at 53.5% majors and 46.5% non-majors. Introduction to Java was the first programming class that 60.4% of the participants had taken, while 39.6% reported taking a programming class previously. Participants who were 21 years old and over made up 53.1% of the population while 46.9% were under the age of 21.

Table 1

Sample Demographic Data

	n	%
Gender		
Male	114	79.2
Female	30	20.8
Classification		
Freshman	12	8.3
Sophomore	40	27.8
Junior	77	53.5
Senior	10	6.9
Graduate	4	2.8
Major		
Computer Science	77	53.5
Non-Computer Science	67	46.5
First Computer Science Class		
Yes	87	60.4
No	57	39.6
Age		
21 and over	77	53.1
Under 21	68	46.9

Note. Sample demographics (n = 144).

Reflective thinking survey. The reflective thinking survey had a 5-point Likert scale that participants used to evaluate themselves on each of the four dimensions of reflection. Each dimension had four questions associated with it, for a total of 16 questions in the survey. The average response value and standard deviation are presented in Table 2.

Table 2

Reflective Thinking Survey Individual Items

Survey Questions	*M*	*SD*
Habitual action		
1. When I am working on some activities, I can do them without thinking about what I am doing.	3.388	1.359
5. In this course we do things so many times that I started doing them without thinking about it.	3.510	1.195
9. As long as I can remember handout material for examinations, I do not have to think too much.	2.612	1.198
13. If I follow what the lecturer says, I do not have to think too much on this course.	2.561	1.167
Understanding		
2. This course requires us to understand concepts taught by the lecturer.	4.347	.994
6. To pass this course you need to understand the content.	4.713	.668
10. I need to understand the material taught by the teacher in order to perform practical tasks.	4.248	1.004
14. In this course you have to continually think about the material you are being taught.	4.287	.887
Reflection		
3. I sometimes question the way others do something and try to think of a better way.	4.176	.969
7. I like to think over what I have been doing and consider alternative ways of doing it.	4.088	.996

(continued)

Table 2 (continued)

Reflective Thinking Survey Individual Items

Survey Questions	*M*	*SD*
11. I often reflect on my actions to see whether I could have improved on what I did.	4.167	.955
15. I often re-appraise my experience so I can learn from it and improve for my next performance.	4.078	.840
Critical Reflection		
4. As a result of this course I have changed the way I look at myself.	2.843	1.280
8. This course has challenged some of my firmly held ideas.	2.735	1.327
12. As a result of this course I have changed my normal way of doing things.	2.804	1.203
16. During this course I discovered faults in what I had previously believed to be right.	3.206	1.129

Internal reliability. Cronbach's Alpha values were calculated for the four different dimensions using the data gathered from this study (Table 3). The resulting scores for reflection and critical reflection were above .70. The scores for habitual action and understanding were below .70.

Table 3

Reliability of Survey of Reflective Thought

Dimension	*M*	*SD*	α
Habitual action	12.071	3.134	0.509
Understanding	17.594	2.397	0.583
Reflection	16.510	2.817	0.737
Critical Reflection	11.588	3.719	0.743

Web-CAT usage. Participant usage data of the Web-CAT software was collected and participants who used the tool were split into low and high usage categories based on the total number of times the participants submitted their code to Web-CAT (Table 4). The median level of submissions was seven, so participants with seven or more submissions were grouped into the high category and those who submitted less than seven times were grouped into the low category.

Table 4

Total usage of Web-CAT

Submission level	*n*
No submission	70
Low (below 7)	24
High (7 and above)	29
Total	123

In addition to the number of times a student submitted to Web-CAT, values were also collected for the final percentage of tests that a student was passing for each project as well as the score (out of 100%) that the Checkstyle tool gave their submission (Table 5). As the semester progressed, the number of students submitting to each project decreased, as did the average number of submissions. The average test pass rate continued to improve throughout the semester, and the average Checkstyle score became much higher at the end as well.

Table 5

Usage of Web-CAT by Project

Project	Total Users	Average number of submissions	Average test pass rate	Average Checkstyle score
3	50	5.62	56.728%	28.400%
4	46	4.43	87.359%	22.848%
5	37	2.86	94.595%	62.054%

Results

The data collected in this study were analyzed according to each research question and their underlying hypotheses and will be addressed individually. The questions were:

1. How does the use of automated software testing influence levels of reflective thought in students compared to students who do not use automated software testing?

2. How does the use of automated software testing influence student performance on introductory computer science programming compared to students who do not use automated software testing?

3. To what degree does reflective thought affect student performance on programming assignments for those who use automated software testing compared to those who do not?

4. How does demographic data influence levels of reflective thought in students both who use and who do not use automated software testing?

5. How does demographic data influence student performance on introductory computer science programming assignments?

6. How does demographic data influence student usage of an automated software-testing environment?

Research Question 1. This research question had four corresponding hypotheses, one for each dimension of the reflective thought survey. Students who used the Web-CAT software were hypothesized to have significantly different levels of habitual action (H_1), understanding (H_2), reflective thought (H_3), and critical reflection (H_4). A one-way MANOVA test was conducted for each hypothesis to determine if there was a difference in levels of reflective thought based on Web-CAT submission level. For H_1, the test was not significant and the null hypothesis was not rejected. For H_2, the test was significant at the $p < .05$ level, with values of $F(2,100) = 3.640$, $p = .030$. Post-hoc analysis performed using Tukey's HSD test revealed that students in the high usage category ($M = 16.571$, $SD = 3.096$) were found to have significantly lower scores in the understanding dimension than those who did not use the tool at all ($M = 17.885$, $SD = 1.896$). The null hypothesis was rejected. For H_3, the test was not significant and the null hypothesis was not rejected. For H_4, the test was not significant and the null hypothesis was not rejected.

Research Question 2. This research question had a single hypothesis, H_5, which stated that average student performance on programming assignments would be higher for those who use an automated software-testing environment than for those who do not. An ANOVA test was conducted to determine if there was a difference in student performance based on Web-CAT submission level. The test was significant at the $p < .05$ level, with values of $F(2, 120) = 5.044$, $p = .008$. The null hypothesis was rejected. Post-hoc analysis performed using Tukey's HSD test revealed that students in the high usage category ($M = 80.390$, $SD = 19.004$) were found to have significantly higher average project scores than those who did not use the tool ($M = 63.962$, $SD = 26.909$). However, the performance of students in the low usage category ($M = 74.083$, $SD = 23.069$) did not differ significantly from those who did not use Web-CAT.

Table 6

Total usage of Web-CAT and Average Student Performance

Submission level	*n*	*M*	*SD*
No submission	70	63.962	26.909
Low (below 7)	24	74.083	23.069
High (7 and above)	29	80.390	19.004
Total	123	69.810	25.349

Research Question 3. This research question had four corresponding hypotheses, one for each dimension of the reflective thought survey. Students who used the Web-CAT software were hypothesized to have a significant relationship between student performance and habitual action (H_6), understanding (H_7), reflective thought (H_8), and critical reflection (H_9). Pearson's product-moment correlation was run for each Web-CAT usage group to determine the relationship between reflective thought and student performance. Correlations were run for students who did not use Web-CAT at all, those who used Web-CAT in the low usage level (from 1 to 6 times), and for those who used Web-CAT seven times or more. For each of these usage levels, the results of the tests indicated that there was no significant correlation between the four variables and performance. With regard to the hypotheses, the tests were not significant for any of the Web-CAT usage levels and therefore the null hypotheses for H_6 through H_9 were not rejected. Additional analysis using multiple regression was desired but not conducted because there was not a linear relationship between the data and so the prerequisites for the test were not met.

Research Question 4. This research question had four corresponding hypotheses which stated that student demographics would not influence self-reported levels of habitual action (H_{10}), understanding (H_{11}), reflection (H_{12}), and critical reflection (H_{13}). Five demographic values were tested: gender, age (under 21 or 21 and over), major (computer science or not), classification, and whether it was their first time in a computer science class or not. A factorial MANOVA was used to test the influence of these five demographic variables on the four self-reported levels on the reflective thought survey.

For habitual action (H_{10}), the results showed that the interaction between first time in a computer science course and age had a significant effect (Pillai's Trace = .121, $F(1,97) = 8.516$, $p = .005$), therefore the null hypothesis was rejected. Older students in a computer science class for the first time had significantly higher levels of self-reported habitual action than younger first time computer science students, while the opposite case was true for students who were not in a computer science class for the first time. Figure 1 shows a graph of this interaction.

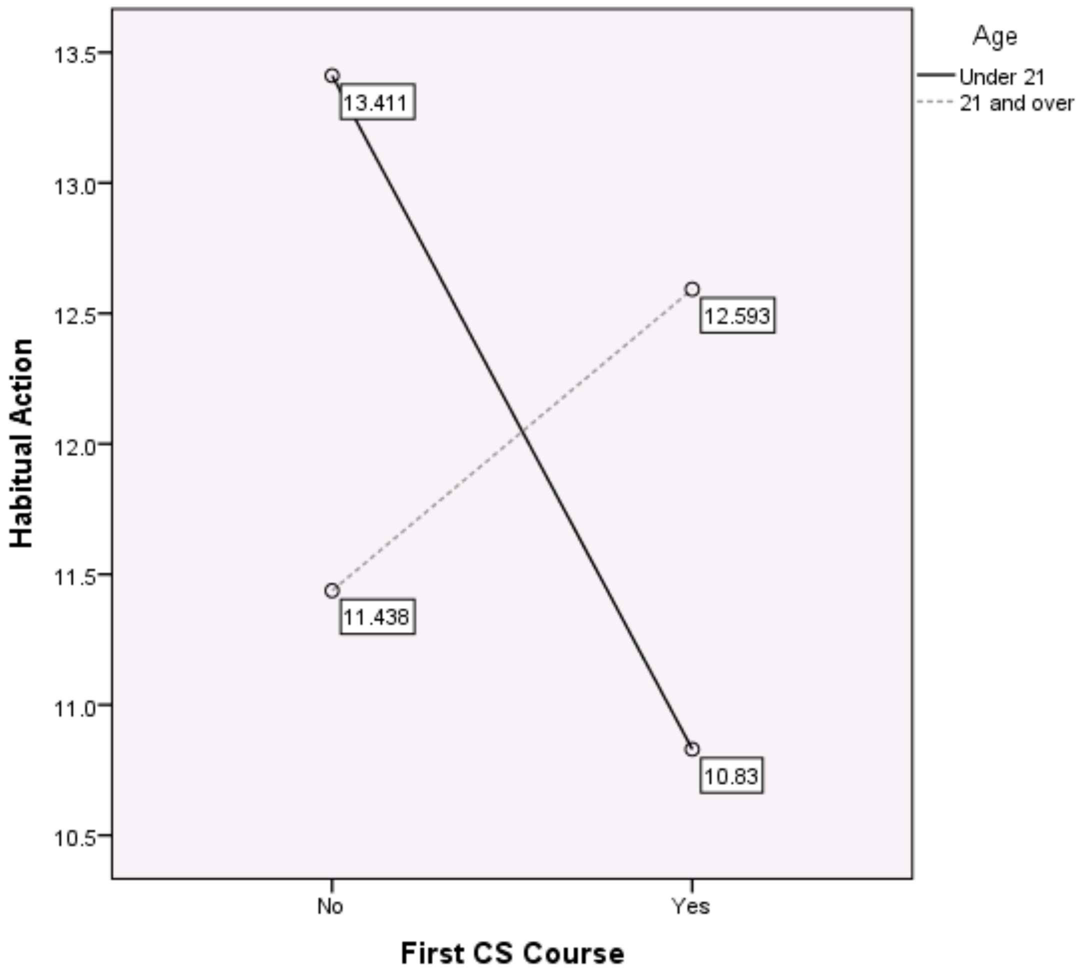


Figure 1. Line graph showing self-reported levels of critical reflection by age and first time in a computer science course.

For understanding (H_{11}), the results showed that the interaction between classification and age had a significant effect (Pillai's Trace = .109, $F(1,97) = 5.495$, $p = .022$), therefore the null hypothesis was rejected. Levels of self-reported understanding were slightly higher for younger sophomores when compared to older sophomores, but younger juniors had dramatically lower levels than older juniors. Figure 2 shows a graph of this interaction.

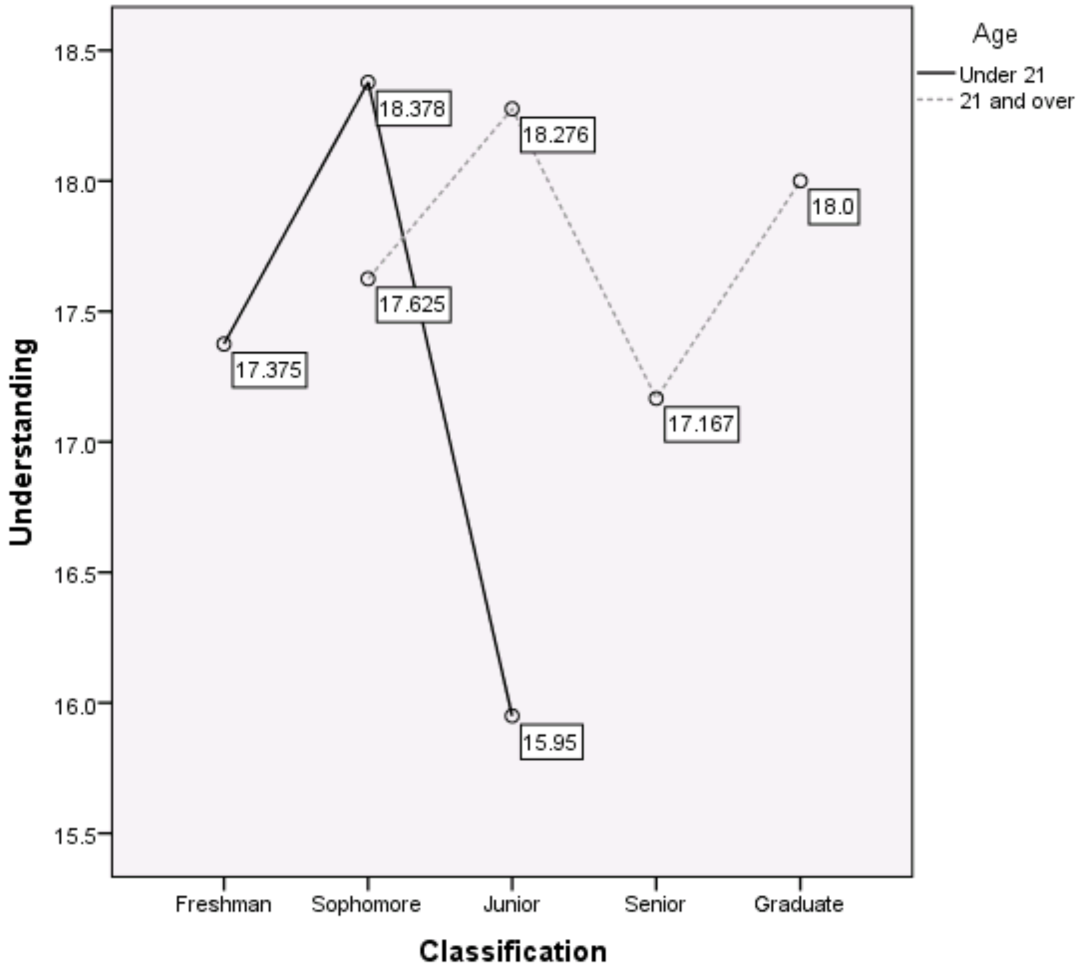


Figure 2. Line graph showing self-reported levels of understanding by age and classification.

For reflection (H_{12}), there was no significant effect and therefore the null hypothesis was not rejected. For critical reflection (H_{13}), the results showed that the interaction between classification and major had a significant effect (Pillai's Trace = .253, *F*(3,97) = 4.759, *p* = .005), therefore the null hypothesis was rejected. Freshman computer science majors had a much higher level of critical reflection than non-majors, while sophomore non-majors had much higher critical reflection than computer science majors. Figure 3 shows a graph of this interaction.

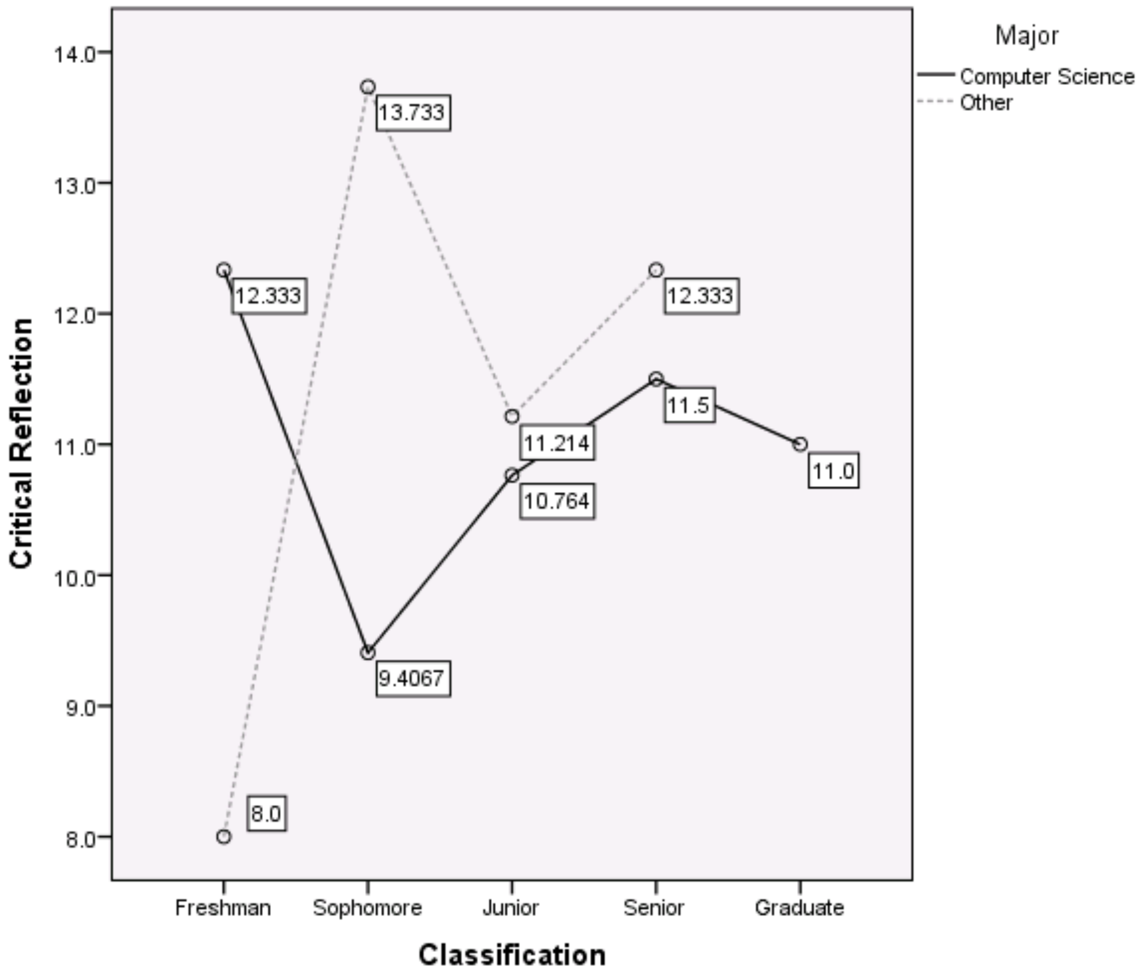


Figure 3. Line graph showing self-reported levels of critical reflection by major and classification.

Research Question 5. This research question had a single hypothesis, H_{14}, which stated that student demographics would not influence average student performance on programming assignments. Five demographic values were tested: gender, age (under 21 or 21 and over), major (computer science or not), classification, and whether it was their first time in a computer science class or not. A factorial MANOVA was used to test the influence of these five demographic variables on the student's average performance on projects 3 through 5. The results showed that there was no significant effect on student performance; therefore the null hypothesis was not rejected.

Research Question 6. This research question had a single hypothesis, H_{15}, which stated that student demographics would not influence student usage of an automated software-testing environment. Five demographic values were tested: gender, age (under 21 or 21 and over), major (computer science or not), classification, and whether it was their first time in a computer science class or not. A factorial MANOVA was used to test the influence of these five demographic variables on the number of overall times a student submitted files to the Web-CAT software. The results showed that there was no significant effect on the number of submissions; therefore the null hypothesis was not rejected.

Chapter Summary

This chapter provided an analysis of the results obtained in this study. There were 144 participants in this study collected from a broad variety of ages, classifications, and fields of study. Because usage of the Web-CAT software was optional, the grouping of the participants was self-selecting and resulted in three usage levels: no usage, low usage (less than seven total submissions), and high usage (seven or more total submissions). These usage data were combined with participant demographic and reflection surveys as well as software project performance data gathered from their instructors to answer the research questions described in this study.

The first research question regarding reflection and Web-CAT usage found significantly lower levels of understanding reported by participants in the high Web-CAT usage category. All other categories of reflection did not have significant results. The second research question involved Web-CAT usage and student performance. In this case, students who were in the high usage category were found to have significantly higher project scores than those who did not use the tool at all. Research question three proposed a relationship between levels of reflection and student performance, but no correlation with student performance was found for any of the dimensions of reflection.

The final three research questions involved investigating the relationships between student demographics and the other variables in this study. The fourth research question examined the relationship between demographics and levels of reflection. Habitual action was significantly affected by age and whether it was the student's first computer science course. Understanding was also significantly affected by age and a student's classification. The top two levels of reflection and critical reflection were not affected by any demographic variables, however. Research question five did not indicate any significant relationships between student demographics and student performance. Similarly, research question six also did not indicate a relationship between student demographics and usage of the Web-CAT software.

CHAPTER V

DISCUSSION

This study was designed to examine the effectiveness of an automated software-testing environment and investigate the possibility that reflection might be associated with performance while using such a tool. In this chapter, the findings of this study will be summarized and discussed with an emphasis on their educational impact and implications for future work.

Discussion of Results

There were five research questions in this study, several of which had multiple hypotheses associated with it. The discussion of results in this section will be organized by research question. A summary of the results for each question will be given, followed by a discussion of these results.

Research Question 1. The first research question explored the effect of automated software testing on reflective thought. This question had four hypotheses that corresponded to each level of reflective thought, H_1 through H_4, and was answered using a one-way MANOVA test.

Findings. Self-reported levels of reflective thought were measured with the reflective thought survey developed by Kember et al. (2000). Responses to this survey were tallied to yield four scores representing levels of reflective or non-reflective thought. Usage of Web-CAT was determined by grouping students into three different categories based on their total usage of Web-CAT. Students with the mean number of submissions (seven) or higher were grouped into a high usage category, while those who submitted code less frequently than that were placed in a low usage category. Students who did not use Web-CAT at all were grouped into a third category of no usage. A one-way MANOVA test was run to see if any of the reflective survey dimensions varied significantly based on Web-CAT usage. The results showed that of the four dimensions of reflective thought, the only one to have significantly different levels based on Web-CAT usage was understanding. A post-hoc test using Tukey's HSD showed that students in the high usage level had significantly lower levels of reported understanding than those in the low usage level or those who did not use the tool at all.

Discussion. Mezirow (1991) discussed types of action as they relate to reflection, which at a high level he separated into non-reflective and reflective action. He further separated non-reflective action into habitual action and thoughtful action, while reflective action was separated into reflective action and premise (or critical) reflection (Mezirow, 1991, pp. 106-110). These four categories were the basis for Kember et al.'s (2000) Questionnaire for Reflective Thinking, which had similar categories of habitual action, understanding, reflection, and critical reflection. Of the four categories, understanding is the only one that differs substantially from Mezirow's.

In their discussion of this category, Kember et al. (2000) begin with Mezirow's (1991) thoughtful action category, stating "thoughtful action can be described as a cognitive process. Much of the 'book learning' which takes place in universities is best classified as thoughtful action" (p. 384). This relationship with learning led Kember et al. (2000) to refer to Bloom's (1984) taxonomy as a way of measuring this type of thoughtful action. Because the scope of this concept was so large, their measurement was narrowed to the comprehension portion of Bloom's (1984) taxonomy. According to Kember et al. (2000):

> [Bloom's] definition of comprehension as "understanding without relating to other situations" captured the distinction we wished to make (which was) an academic type of learning in which the student might reach an understanding of a concept without reflecting upon its significance in personal or practical situations (p. 384).

Kember et al. (2000) observed a positive correlation between understanding and both reflection and critical reflection. They stated, “Students who engage in either form of reflection may also have a tendency to study for understanding, particularly in more theoretical parts of a course, which have less obvious relationships to practice” (Kember et al., 2000, p. 389). In this study, the understanding dimension did not significantly correlate to any other dimension. Instead, students who used Web-CAT had significantly lower levels of self-reported understanding.

A decrease in a student’s self-reported levels of understanding means that these students agreed less with the statements that they were required to study for understanding in the course. Yet these students also used Web-CAT more, and as we have observed in the results for Research Question 2 they also showed higher performance on software projects. Kember et al. (2000) suggested that understanding was related to theoretical understanding in a course, such as performance on tests and quizzes. This measurement of understanding may not have had much of a connection with practical applications such as software projects, however.

Kember et al. (2000) refer to the short time of a university course as a possible reason why both habitual action and critical reflection showed lower mean scores than the other two dimensions. Lower mean scores for these two dimensions were also observed in this study. The limited time constraints of this study may have been a reason why reflection levels failed to change significantly during the observation. Future studies may benefit from either a longer observation window during a semester or even span multiple semesters to give reflection a chance to occur.

The failure to observe a measured change in levels of reflection could also be a confirmation of Edwards’ (2004) theory that true reflection-in-action would only occur when a student is "given the responsibility of demonstrating the correctness of his or her own code" by writing their own test cases for Web-CAT. Marrero and Settle (2005) also noted that requiring students to write test cases forced them to think more about how the software they were writing was supposed to be used. Test-driven design requires students to be involved in experimentation and reflective thought during the iterative process of software testing and design. These actions align themselves very well with the modes of learning referred to as active experimentation and reflective observation in Kolb's (1984) LSI, as well as Schön's (1983) theory of reflection-in-action. Given the practical constraints of this study it was not possible to require students to submit such test cases, but this remains a useful question to be answered in future work.

Conclusion. Significantly different levels of reflective thought were not observed in this study for students who used Web-CAT at any level. A decrease in studying for understanding was observed for students who used Web-CAT at a high level, but this was not considered reflective thought and could be attributed to differences in how these students studied for the theoretical portions of the course or other factors not being measured in this study.

Research Question 2. The second research question explored the effect of automated software testing on average student performance on programming assignments. This question had one hypothesis (H_5) and was answered using a one-way ANOVA test.

Findings. Average student performance was computed by averaging student scores for projects three, four and five. Only students with scores for all three projects were used for this test. The Web-CAT usage categories of none, low, and high usage that were created for research question one were used in this question as well. A one-way ANOVA test was run and the results showed that average student performance differed significantly based on levels of Web-CAT usage. Post-hoc analysis was done using Tukey's HSD test, and the results showed that students in the high usage level had significantly higher project scores than those who did not use Web-CAT at all. Students in the low usage level, however, did not have significantly different project scores than those who did not use Web-CAT.

Discussion. Many of the existing works on automated grading describe the introduction of new grading tools but do not describe the effects that such tools have had on student performance (Higgins et al., 2003; Jackson & Usher, 1997; Joy et al., 2005). Edwards' (2004) study, however, did compare the performance of students who used an earlier automated grading tool called Curator with those who were required to write their own test cases using Web-CAT. Students using Web-CAT in Edwards' (2004) study performed significantly better when averaging the scores of four programming assignments. The use of TDD has not been consistently shown to improve student performance, however. Marrero and Settle (2005) studied the effects of TDD on student performance and found that student performance did not significantly increase when students were required to write test cases.

This study did not require students to write their own test cases. Rather, it relied on test cases written by the researcher to evaluate whether student submitted code met the requirements of the assignment. Edwards (2003) claimed that the problem with prior automated grading systems without student test cases (such as Curator) was that "students focus on output correctness first and foremost…due to the fact that the most immediate feedback students receive is on output correctness" (p. 149). This may have been true in this study as well, though the presentation of an evaluation opportunity to students will often result in their being interested in seeing how well they perform. Students who used Web-CAT also had their code style and formatting evaluated by a Checkstyle tool. Over time it was shown that the mean values for both percentage of JUnit tests passed and Checkstyle evaluation scores improved.

Conclusion. Previous research has involved two factors: introduction of student-written test cases and student usage of automated testing tools. Edwards (2004) used automated testing tools throughout his study, but observed an increase in student performance when student-written tests were used. Marrero and Settle (2005) did not use automated testing, but saw no improvement when student-written tests were introduced. This study looked at a third combination of these factors by not using student-written test cases but introducing automated software testing. Though student-written tests were not used in this study, students who used Web-CAT frequently saw their average project scores improve significantly.

Research Question 3. The third research question explored the effect of reflective thought on student performance when Web-CAT usage was taken into account. There were four hypotheses associated with each level of reflective thought, H_6 through H_9, and analysis was performed using Pearson's product-moment correlation. The variables used to answer this research question (self-reported levels of reflective thought, average student performance, and Web-CAT usage category) were the same as those defined previously in research questions one and two. A Pearson product-moment correlation was run for each level of Web-CAT usage, and all three tests indicated that there was no correlation between the four levels of reflective thought and average student performance.

As mentioned in the discussion of research question one, levels of reflection failed to vary significantly among the different levels of Web-CAT usage. The lack of change in reflection may have been due to the limited time available during the semester or because students were not required to write their own test cases. Even though student performance has been shown to vary with Web-CAT usage, there was no observed relationship between variations in student-reported levels of reflection and average project performance for students in the same Web-CAT usage level.

Research Question 4. The fourth research question examined whether or not student demographic data influenced student self-reported levels of reflection. There were four hypotheses associated with each level of reflective thought, H_{10} through H_{13}, and a factorial MANOVA was used to test for both main effects and interaction effects of the demographic variables on levels of reflection.

Findings. There were five independent variables in this research question that were collectively known as student demographics. These variables were: gender, age (under 21 or 21 and over), major (computer science or not), classification, and whether it was their first time in a computer science class. The four dependent variables that represent levels of student-reported reflective thought have been previously discussed in research questions one and three.

Because of the number of variables used in the factorial design used to test these hypotheses, the resulting list of effects was quite long. Most of the interactions were not significant, but three of the four reflection levels had a specific interaction that affected it. Habitual action (H_{10}) was shown to be affected by the interaction of whether it was the student's first time in a computer science course and their age. Older first-time computer science students had higher levels of habitual action as well as younger computer science students with previous experience (Figure 1). Understanding (H_{11}) was affected by interaction of classification and age. Studying for understanding happened most with Sophomores under 21 and Juniors 21 and over, while Juniors under 21 reported much lower levels of understanding (Figure 2). None of the demographic variables significantly affected reflection (H_{12}), but critical reflection (H_{13}) was significantly affected by the interaction of whether the student was a computer science major or not and their age. Freshman non-majors had dramatically lower levels of critical reflection than freshman computer science majors, while sophomore computer science majors reported levels that were both higher than non-majors and also the highest levels reported (Figure 3).

Discussion. Demographic values were examined for their effect on the primary variables of reflection in this study so as to exclude them as potentially confounding variables. Though no single variable was shown to affect reflection, several combinations were associated with significantly different reflection levels. Though the effects of these demographics on reflection levels are interesting, the existing literature does not contain observations to confirm or explain these specific interactions. Further research is needed into the effects of demographic variables such as first time in a computer science course, classification, and age on self-reported levels of student reflection, specifically using the instrument developed by Kember et al. (2000).

Research Question 5. The fifth research question examined whether or not student demographic data influenced average student performance on programming assignments. There was a single hypothesis, H_{14}, and a factorial design was used to test for both main effects and interaction effects of the demographic variables on student performance. No significant effects were found to show an effect of demographics on student performance. The variations in student performance observed in research question 2 cannot be attributed to any of these demographic variables.

Research Question 6. The sixth research question examined whether or not student demographic data influenced student usage of the Web-CAT software testing environment. There was a single hypothesis, H_{15}, and a factorial design was used to test for both main effects and interaction effects of the demographic variables on Web-CAT usage. No significant effects were found to show an effect of demographics on Web-CAT usage.

Implications of the Study

The goal of this study has been to examine the effectiveness of automated software-testing tools like Web-CAT in introductory programming courses. Do automated software-testing tools improve student performance, even if test-driven design methods aren't used? If so, could Edwards' (2004) application of Schön's (1983) reflection-in-action theory be linked to Web-CAT usage and explain some of the improvements in student performance? In this study, average student project performance has been shown to increase with high levels of Web-CAT usage. Reflection, however, has not been shown to have any connection with Web-CAT usage or student performance. Even though the link to reflection was not verified in this study, there are still several important implications regarding the academic benefits of automated software-testing tools. Several stakeholders were previously identified as potentially benefitting from the results of this study. The implications for each of them will be discussed individually in this section.

It is important to note that when adoption of an automated software-testing tool is discussed, Web-CAT is used as the primary example. This is only because this tool was used for this study and is therefore the one that the researcher has had the most experience with. There are several other tools available that may meet the stakeholders' needs as well or better, and the stakeholders are advised to investigate each of the tools available to see what might be the best fit for their application.

Instructors. Adoption of an automated software-testing tool is not a trivial decision to make. It requires a restructuring of curriculum to make assignments more testable, and test cases must be written for each assignment in order to evaluate student submissions. The automated testing tool must be installed and maintained. Students must be assigned accounts for Web-CAT, taught how to use the software, and have their technical support questions answered. Much of this work will fall to the instructor who first has the vision to adopt this tool.

Though the workload involved in carrying out such a project is significant, the benefits from doing so are significant as well. Student project performance has been shown to increase in both Edwards (2004) and in this study. Also, much of the effort expended in setting up an automated grading tool will be recouped after it is set up. Once an instructor has gone through the additional effort of creating testable assignments and developing a useful suite of test cases, the assignments could then be used in future semesters with little rework. Teaching assistants, if available, can be used to support the grading tool and do much of the basic syntax and style grading on their own. All of this infrastructure would allow an instructor to spend more time advising students on coding style, software design, and the quality of their work.

For maximum benefit it is recommended that instructors require the usage of Web-CAT as the single central place for code submission, testing, grading and feedback. If this approach were taken, a trail of progress and feedback would be available for each student's project. Instructors would then have a single repository containing multiple revisions of each student's code. This would allow them to see each student's progress and examine who is having trouble with concepts or is lagging behind. The latest version of a student's code would be readily accessible, which could help in meetings during office hours or with email correspondence. These progress trails would also help reduce plagiarism because instructors would have more evidence to examine in addition to a final submission. Many of these benefits could also be realized if any version control system were adopted, but the integrated nature of a tool like Web-

CAT means that the students and instructors can achieve all these goals by learning and using one single tool.

Computer science departments. As previously described, setting up an automated grading tool requires a lot of effort and planning. It would not be feasible for each instructor to set up their own implementation of automated grading, each with their own server. Computer science departments can encourage and coordinate adoption of these tools at a departmental level, making them available to instructors who are interested in using them. Departments can also make server and technical resources available towards this goal, establishing a central installation of a tool like Web-CAT to be used by all students, even from multiple classes. In exchange for this investment, departments will have empowered instructors with tools that will help them teach test-driven programming and grade their projects more quickly and accurately.

Students. Students may have the least influence on the adoption of a tool like Web-CAT, but they would certainly benefit the most. If instructors or departments adopt this type of software, students may experience improved project performance because they will spend more time thinking about, testing, and refining their software. Students will receive instant feedback not only on the correctness of their software but also on the formatting and style of the code. Their code will be backed up on the server, and they will always be able to log in and see how their code is progressing from any web browser.

Limitations of the Study

Kember et al. (2000) suggested using their questionnaire for reflective thinking in a repeated measures design, administering it at the beginning and at the end of the courses they were testing. They stated that "any changes to reflective thinking can then be reasonably attributed to the course and it's teaching and learning environment" (Kember et al., 2000). In this study the survey was administered only once, so it was not possible to measure changes in a person's self-reported levels of reflection. Because the effect on the students may have been a small one, it is possible that a student's tendency to respond positively or negatively to the survey may have overshadowed any changes in reflection that the use of the Web-CAT tool might have generated.

The levels of internal reliability measured while using Kember et al.'s (2000) questionnaire were quite low for the dimensions that measured non-reflective thought. The Chronbach's alpha values calculated for habitual action (0.509) and understanding (0.583) were very poor compared to the traditional threshold of .70 (Nunnally & Bernstein, 1994) and were lower than those found in the instrument author's previous studies (Kember et al., 2000; Leung & Kember, 2003). This may mean that the students did not respond to each collection of questions as consistently as students in the previous studies, or it could be that the limited number of questions measuring one dimension was a limiting factor on internal reliability.

The two categories measuring reflection had much higher alpha values, however. Values for reflection (0.737) and critical reflection (0.743) were above the traditional .70 threshold (Nunnally & Bernstein, 1994) and were higher than those found in the instrument author's previous studies (Kember et al., 2000; Leung & Kember, 2003). This inconsistency with the reflective thinking survey could have been ameliorated by the use of repeated measures as mentioned previously or by the use of additional instruments that measured similar concepts. Leung and Kember (2003) used the Study Process Questionnaire (Biggs, 1987) alongside their reflective thought survey and found favorable relationships between them, encouraging the use of the two frameworks together in future studies.

The nature of the introductory Java course being studied meant that student reliability in completing assignments or turning them in promptly was often sporadic. The instructors participating in this study usually offered students the opportunity to “drop” their lowest grade. This resulted in several low or zero grades reported for projects three through five, yet students did well in the course because they did optional work or otherwise excelled later in the semester. In spite of all this, if a grade of zero was reported for a student during the period being studied it was used to calculate their grade average for this study. The frequent occurrence of these zero grades may have affected the measurement of student performance in such a way that it did not reflect their total achievement for the semester. A longer study involving the use of Web-CAT for the whole semester or even multiple semesters would have allowed more data points on performance to be gathered and perhaps a more accurate measurement to be made.

Recommendations for Further Research

This study involved the optional use of Web-CAT as a supplementary tool for a single semester in a course that did not include student-written tests. Additional research is needed to see if variations in any of these factors can be further shown to influence student performance, and if self-reported levels of student reflection can be associated with these changes. What if Web-CAT was a mandatory tool for a class? In a mandatory situation a student would have to submit their code at least once. What level of student usage of Web-CAT would have to happen before a student began to see benefits in performance?

The benefit of student-written test cases in an automated testing environment is another topic that requires future research. Marrero and Settle (2005) found that student-written tests alone do not affect student performance. However, they were not using an automated software-testing tool. Would student-written tests make a difference in levels of reflection as Edwards (2004) theorized?

The setting for this study involved classes that were taught in person. However, many universities offer introductory computer science courses entirely online. If a similar study were conducted with online-only students, would similar increases in student performance be observed? Perhaps course delivery method could be accounted for by conducting an experiment that involved large enough numbers of both in-person and online participants so that it would be possible to control for any effects that online course delivery could introduce. It would also be interesting to ask online-only students what problems were solved or introduced when they were asked to use an automated software-testing tool.

The demographic data collected in this study was interesting because of the observed relationships to several of the dimensions on the reflection survey. However, there could also have been many more factors influencing reflection than just usage of Web-CAT. Biggs (1987) describes approaches to learning that may account for some of the variation in observed levels of reflection. In future research studies it may be useful to administer Biggs' (1987) Study Process Questionnaire (SPQ) in order to group students with similar approaches to learning. These groups could then be examined to see if those who use automated testing software report higher levels of reflection or have higher average project scores than those who do not.

Further research is also needed to examine the issue of automated testing and reflection at higher levels and for longer periods of time. This study only focused on three projects in the middle of a semester. How do levels of reflection affect overall student performance, including exams and other grades? Is there a connection between Web-CAT usage and overall class performance? As previously suggested, a study involving the same group of students using

Web-CAT over multiple semesters of computer science classes would allow repeated measurements of reflection to be taken and perhaps allow for more variation. These higher-level investigations may provide a clearer picture of the relationships between reflection and student performance in an automated testing environment. These students would also be more likely to be computer science majors, thereby more accurately reflecting the target population.

Conclusion

Automated software-testing tools like those described by Edwards (2003) and Spacco (2006) introduce a level of automation to introductory computer science classes that has been missing from many classrooms. They allow both instructors and students to spend more time focusing on quality. Instructors can allow their test suite to examine validity and edge-case testing while freeing up time for them to examine student code for things like style and formatting. Students have an instructor test suite to guide them, and if they desire to write their own tests they can allow the tests to help them bridge the gap between requirements and implementation. This study has shown that high usage of Web-CAT is associated with an improvement in average student performance, even without the use of student-written tests.

Edwards (2004) makes a strong case for asking students to write test cases, even during their very first computer science class. The curriculum or other limitations may prevent instructors from making such a change in their courses. Rather than not using automated grading at all, instructors should seek to integrate such technology where possible. Students should be given the benefit of receiving feedback on their code's accuracy and formatting immediately without having to wait until it is graded. Measures such as masking the test data and obscuring the types of test being used can help counteract "coding for the test" behavior. The benefits of TDD are acknowledged, but student-written tests should be a goal and not a starting point.

REFERENCES

Barriocanal, E. G., Urban, M. S., Cuevas, I. A., & Perez, P. D. (2002). An experience in integrating automated unit testing practices in an introductory programming course. *SIGCSE Bulletin, 34*(4), 125-128. doi:10.1145/820127.820183

Beck, K. (2001). Aim, fire. *IEEE Software, 18*(5), 87-89. doi:10.1109/52.951502

Beck, K. (2003). *Test-driven development by example*. Boston, MA: Addison-Wesley.

Biggs, J. B. (1987). *Student approaches to learning and studying*. Melbourne, Australia: Australian Council for Educational Research.

Bloom, B. S. (1984). *Taxonomy of educational objectives, Book I: Cognitive domain*. New York, NY: Longman.

Boud, D., Keogh, R., & Walker, D. (1985). Promoting reflection in learning: A model. In D. Boud, R. Keogh, & D. Walker (Eds.), *Reflection: Turning experience into learning* (pp. 18-40). London, England: Kogan Page.

Christensen, H. B. (2003, June). *Systematic testing should not be a topic in the computer science curriculum!* Paper presented the 8th annual conference on Innovation and technology in computer science education, Thessaloniki, Greece.

Cutler, B., Cook, P., & Young, J. (1989, February). *The empowerment of preservice teachers through reflective teaching*. Paper presented at the annual meeting of the Association of Teacher Educators, St Louis, MO. Abstract retrieved from ERIC database. (ED325473)

Desai, C., Janzen, D., & Savage, K. (2008). A survey of evidence for test-driven development in academia. *Inroads – SIGCSE Bulletin, 40*(2), 97-101. doi:10.1145/1383602.1383644

Dewey, J. (1910). *How we think*. Boston, MA: Heath & Co.

Dewey, J. (1938). *Experience and education.* New York, NY: Collier MacMillan Publishers.

Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming. *ACM Journal of Educational Resources in Computing, 5*(3), 1-14. doi:10.1145/1163405.1163409

Edwards, S. H. (2003, October). *Rethinking computer science education from a test-first perspective.* Paper presented at the 18th ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications, Anaheim, CA.

Edwards, S. H. (2004, March). *Using software testing to move students from trial-and-error to reflection-in-action*. Paper presented at the SIGCSE Technical Symposium on Computer Science Education, Norfolk, VA.

Erdogmus, H., Maurizo, M., & Torchiano, M. (2005). On the effectiveness of the test-first approach to programming. *IEEE Transactions on Software Engineering, 31*(3), 226-237. doi:10.1109/TSE.2005.37

Hatton, N., & Smith, D. (1995). Reflection in teacher education: Towards definition and implementation. *Teaching & Teacher Education, 11*(1), 33-49. doi:10.1016/0742-051X(94)00012-U

Higgins, C., Hegazy, T., Symeonidis, P., & Tsintsifas, A. (2003). The CourseMarker CBA system: Improvements over Ceilidh. *Education and Information Technologies, 8*(3), 287-304. doi:10.1023/A:1026364126982

Hollingsworth, J. (1960). Automatic graders for programming classes. *Communications of the ACM, 3*(10), 528-529. doi:10.1145/367415.367422

Huang, L., & Holcombe, M. (2008). Empirical investigation towards the effectiveness of Test First Programming. *Information and Software Technology, 51*, 182-194. doi:10.1016/j.infsof.2008.03.007

Jackson, D., & Usher, M. (1997). Grading student programs using ASSYST. *ACM SIGCSE Bulletin, 29*(1), 335-339. doi:10.1145/268085.268210

Janzen, D., & Saiedian, H. (2005). Test-driven development: Concepts, taxonomy, and future direction. *Computer, 38*(9)*,* 43-50. doi:10.1109/MC.2005.314

Jarvis, P. (1987). *Adult learning in the social context.* London, England: Croon Helm, Ltd.

Joy, M., Griffiths, N., & Boyatt, R. (2005). The BOSS online submission and assessment system. *ACM Journal on Educational Resources in Computing*, *5*(3), 1-28. doi:10.1145/1163405.1163407

Keefe, K., Sheard, J., & Dick, M. (2006, January). Adopting XP practices for teaching object oriented programming. In D. Tolhurst, & S. Mann, (Eds.), *Computing Education 2006. Proceedings of the Eighth Australasian Computing Education Conference (ACE2006), 52,* 91-100. Retrieved from: http://crpit.com/confpapers/CRPITV52Keefe.pdf

Kember, D., Leung, D., Jones, A., Loke, A., McKay, J., Sinclair, K., … Yeung, E. (2000). Development of a questionnaire to measure the level of reflective thinking. *Assessment and Evaluation in Higher Education*, *25*(4), 381-395. doi:10.1080/026029300449272

Kluger, A. N., & DeNisi, A. (1996). The effects of feedback interventions on performance: A historical review, a meta-analysis, and a preliminary feedback intervention theory. *Psychological Bulletin*, *119*(2), 254-284. doi:10.1037/0033-2909.119.2.254

Kolb, D. A. (1984). *Experiential learning: Experience as the source of learning and development.* Englewood Cliffs, NJ: Prentice-Hall.

Kollanus, S. (2010, September). *Test-driven development - still a promising approach?* Paper presented at the Seventh International Conference on the Quality of Information and Communications Technology, Porto, Portugal.

Langer, E. J. (1989). *Mindfulness*. Reading, MA: Addison-Wesley Publishing.

Leung, D., & Kember, D. (2003). The relationship between approaches to learning and reflection upon practice. *Educational Psychology, 23*(1), 61-71. doi:10.1080/01443410303221

Leska, C. (2004). Testing across the curriculum: Square one! *Journal of Computing Sciences in Colleges, 19*(5), 163-169. Retrieved from https://www.ccsc.org/publications/pubsJournal.htm

Lewin, K. (1946). Action research and minority problems. *Journal of Social Issues, 2*(4), 34-46. doi:10.1111/j.1540-4560.1946.tb02295.x

Mann, K., Gordon, J., & MacLeod, A. (2009). Reflection and reflective practice in health professions education: A systematic review. *Advances in Health Science Education, 14*(4), 595-621. doi:10.1007/s10459-007-9090-2

Marrero, W., & Settle, A. (2005). Testing first: Emphasizing testing in early programming courses. *ACM SIGCSE Bulletin, 37*(3), 4-8. doi:10.1145/1151954.1067451

Mezirow, J. (1990). *Fostering critical reflection in adulthood*. San Francisco, CA: Jossey-Bass Publishers.

Mezirow, J. (1991). *Transformative dimensions of adult learning*. San Francisco, CA: Jossey-Bass Publishers.

Muller, M. M., & Hagner, O. (2002). Experiment about test-first programming. *Software, IEE Proceedings, 149*(5), 131-136. doi:10.1049/ip-sen:20020540

Nunnally, J. C., & Bernstein, I. H. (1994). *Psychometric Theory* (3rd ed.). New York, NY: McGraw-Hill.

Piaget, J. (1928) *Judgment and reasoning in the child* (M. Warden, Trans.). London, England: Routledge & Kegan Paul Ltd.

Redmond, B. (2006) *Reflection in action: Developing reflective practice in health and social services*. Hampshire, England: Ashgate Publishing.

Reek, K. A. (1989). The TRY system – or – how to avoid testing student programs. *ACM SIGCSE Bulletin, 21*(1), 112-116. doi:10.1145/65293.71198

Schmitt, N. (1996). Uses and abuses of coefficient alpha. *Psychological Assessment, 8*(4), 350-353. doi:10.1037/1040-3590.8.4.350

Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. New York, NY: Basic Books.

Schön, D. A. (1987). *Educating the reflective practitioner*. San Francisco, CA: Jossey-Bass Publishers.

Spacco, J. W. (2006). *Marmoset: A programming project assignment framework to improve the feedback cycle for students, faculty and researchers*. (Doctoral dissertation). Retrieved from ProQuest Dissertations and Theses database. (UMI No. 3241457)

Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J. K., & Padua-Perez, N. (2006). Experiences with Marmoset. *ACM SIGCSE Bulletin, 38*(3), 13-17. doi:10.1145/1140124.1140131

Spacco, J., & Pugh, W. (2006, October). *Helping students appreciate test-driven development (TDD)*. Paper presented at the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications, Portland, OR.

Skinner, B. F. (1935). Two types of conditioned reflex and a pseudo type. *Journal of General Psychology, 12*(1), 66-77. doi:10.1080/00221309.1935.9920088

Thorndike, E. L. (1998). Animal intelligence: An experimental study of the associative processes in animals. *American Psychologist*, *53*(10), 1125-1127. doi:10.1037/0003-066X.53.10.1125 (Original work published 1898)

University of West Florida. (2014). *Facts & figures: 2013-2014 academic year*. Retrieved from http://uwf.edu/about/at-a-glance/facts--figures/

Wolsey, T. D. (2008). Efficacy of instructor feedback on written work in an online program. *International Journal on E-Learning, 7*(2), 311-329. Retrieved from http://www.aace.org/pubs/ijel/

APPENDICES

Appendix A

Institutional Review Board Approval Letters

University of West Florida

Research and Sponsored Programs
11000 University Parkway, Bldg. 11
Pensacola, FL 32514-5750

Mr. Evorell Fridge
400 Bobwhite Dr.
Pensacola, FL 32514

January 02, 2013

Dear Mr. Fridge:

The Institutional Review Board (IRB) for Human Research Participants Protection has completed its review of your proposal titled "2013-063, An Investigation of the Impact of Automated Software Testing Tools on Reflective Thinking and Student Perfiomance to Introductory Computer Science Programming Assignments," as it relates to the protection of human participants used in research, and granted approval for you to proceed with your study on 01-07-2013. As a research investigator, please be aware of the following:

* You will immediately report to the IRB any injuries or other unanticipated problems involving risks to human participants.

* You acknowledge and accept your responsibility for protecting the rights and welfare of human research participants and for complying with all parts of 45 CFR Part 46, the UWF IRB Policy and Procedures, and the decisions of the IRB. You may view these documents on the Research and Sponsored Programs web page at http://www.research.uwf.edu/internal. You acknowledge completion of the IRB ethical training requirements for researchers as attested in the IRB application.

* You will ensure that legally effective informed consent is obtained and documented. If written consent is required, the consent form must be signed by the participant or the participant's legally authorized representative. A copy is to be given to the person signing the form and a copy kept for your file.

* You will promptly report any proposed changes in previously approved human participant research activities to Research and Sponsored Programs. The proposed changes will not be initiated without IRB review and approval, except where necessary to eliminate apparent immediate hazards to the participants.

* **You are responsible for reporting progress of approved research to Research and Sponsored Programs at the end of the project period 08-09-2013. If the data phase of your project continues beyond the approved end date, you must receive an extension approval from the IRB.**

Good luck in your research endeavors. If you have any questions or need assistance, please contact Research and Sponsored Programs at 850-857-6378 or irb@uwf.edu.

Sincerely,

Richard S. Podemski

Dr. Richard S. Podemski, Associate
Vice President for Research
And Dean of the Graduate School

Carla J. Thompson

Dr. Carla Thompson, Chair
IRB for the Protection of Human
Research Participants

CC: Sikha Bagui, William Evans

Phone 850.474.2824 Fax 850.474.2802
Web research.uwf.edu
An Equal Opportunity/Equal Access/Affirmative Action Employer

University of West Florida
FACULTY & STAFF

E. L. Fridge <efridge@uwf.edu>

Re: Project Approval Notification

1 message

Research and Sponsored Programs [irb] <irb@uwf.edu> Tue, May 28, 2013 at 11:33 AM
To: "E.L. Fridge" <efridge@uwf.edu>

Hi Mr. Fridge,

Your request for extension through 12/14/13 is approved. Please proceed with your research according to the protocol.

Thank you,

On Mon, May 20, 2013 at 11:20 AM, E.L. Fridge <efridge@uwf.edu> wrote:

Hello,

I would like to request an extension of data collection for this study through the Fall 2013 Semester (12/14/2013). I had originally planned to collect data for the Spring and Summer, but I will need to collect data through the Fall as well in order to get a sufficient number of participants. I was not able to find an extension request form on your website, but I spoke with Beth Moulder who told me just to email in my request.

Thanks,

E.L. Fridge

On 1/2/2013 10:48 AM, irb@uwf.edu wrote:

Hello Mr. Fridge,

The Institutional Review Board for Human Research Participants Protection has completed its review of your proposal titled "**2013-063, An Investigation of the Impact of Automated Software Testing Tools on Reflective Thinking and Student Perfiomance to Introductory Computer Science Programming Assignments**".
Your approval is attached.

Cheryl K. Allen, CRA

Research Integrity Coordinator
Research and Sponsored Programs
The University of West Florida
11000 University Parkway
Building 11, Room 113
Pensacola, FL 32514-5750
850-857-6378
FAX: 850-474-2082

Appendix B

Informed Consent Form

INFORMED CONSENT

Title of Research: An Investigation of the Impact of Automated Software Testing on Reflective Thinking and Student Performance in Introductory Computer Science Programming Assignments

I. Federal and university regulations require us to obtain signed consent for participation in research involving human participants. After reading the statements in section II through IV below, please indicate your consent by signing and dating this form.

II. **Statement of Procedure:** Thank you for your interest in this research project being conducted by the staff members of The University of West Florida. By this time, one of the investigators should have described the procedures for you in detail. The purpose of this study is to investigate the impact of automated software testing tools on student reflective thinking and performance. You will find a summary of the major aspects of the study being described below, including the risks and benefits of participating. Carefully read the information provided below. If you wish to participate in this study, sign your name and write the date. Any information you provide to us will be kept in strict confidence. If you have any questions or concerns regarding this project, please contact Evorell Fridge in the Computer Science Department at The University of West Florida at (850) 474-2046 or by email at efridge@uwf.edu.

I understand that:

1) My project grades for this class will be collected and used for data analysis.
2) I will be asked to complete a short demographic survey.
3) After the semester is over, I will be asked to complete a survey about my thinking in this course.
4) I may be chosen to use an automated software-testing tool to help evaluate my computer coding projects. This tool will only be offered on specific assignments, and I must still submit my finished work using the method specified by my instructor.
5) I may discontinue participation in this study at any time without penalty.

III. **Risks, Benefits, and Payments:**

1) There are no physical risks associated with this study.
2) Participants in this study will have the option to be entered into a drawing to receive a $50 Wal-Mart gift card.

IV. **Statement of Consent:** I certify that I have read and fully understand the Statement of Procedure given above and agree to participate research project described therein. Permission is given voluntarily and without coercion or undue influence. It is understood that I may discontinue participation at any time without penalty or loss of any benefits to which I may otherwise be entitled. I will be provided a copy of this consent form.

______________________________	______________________
Type/Print Participant's Name	Date

____
Participant's Signature

Appendix C

Demographic Survey

1. Year in school:
    a. Freshman
    b. Sophomore
    c. Junior
    d. Senior
    e. Graduate

2. Gender:
    a. Male
    b. Female

3. Is this the first Computer Science / programming class you have taken?
    a. Yes
    b. No

4. My Major is:
    a. Computer Science
    b. Another technology-related major
    c. Other

5. State your official major (Computer Science, Information Technology, Engineering, Math, Physics, etc.):

    ____

6. Age: ______________

7. Email Address: __________________________

8. I wish to receive a copy of the results of this study.
    a. Yes
    b. No

Appendix D

Reflective Thinking Survey

UWF email address: ___________________________________

This is NOT a test. There are no 'right' or 'wrong' responses to the statements that follow. A response is only 'right' if it reflects your *personal* reaction, and the *strength* of your reaction, as accurately as possible. Please **circle** the appropriate letter to indicate your level of agreement with statements about your actions and thinking in this course.

A—definitely agree
B—agree with reservation
C—only to be used if a definite answer is not possible
D—disagree with reservation
E—definitely disagree

	Agree				*Disagree*
1) When I am working on some activities, I can do them without thinking about what I am doing	A	B	C	D	E
2) This course requires us to understand concepts taught by the lecturer	A	B	C	D	E
3) I sometimes question the way others do something and try to think of a better way	A	B	C	D	E
4) As a result of this course I have changed the way I look at myself	A	B	C	D	E
5) In this course we do things so many times that I started to do them without thinking about it	A	B	C	D	E
6) To pass this course you need to understand the content	A	B	C	D	E
7) I like to think over what I have been doing and consider alternative ways of doing it	A	B	C	D	E
8) This course has challenged some of my firmly held ideas	A	B	C	D	E
9) As long as I can remember handout material for examinations, I do not have to think too much	A	B	C	D	E
10) I need to understand the material taught by the lecturer in order to perform practical tasks	A	B	C	D	E
11) I often reflect on my actions to see whether I could have improved on what I did	A	B	C	D	E
12) As a result of this course I have changed my normal way of doing things	A	B	C	D	E
13) If I follow what the lecturer says, I do not have to think too much on this course	A	B	C	D	E
14) In this course you have to continually think about the material you are being taught	A	B	C	D	E
15) I often re-appraise my experience so I can learn from it and improve my next performance	A	B	C	D	E
16) During this course I discovered faults in what I had previously believed to be right	A	B	C	D	E

UWF email address: ____________________________________

1) If you were asked to use the WebCat software for this study, which of these statements reflects your level of participation?

a. I successfully used the software to evaluate my code

b. I tried to use the software but was unsuccessful

c. I did not attempt to use the WebCat software

2) If you were asked to use the WebCat software for this study, please rate how difficult it was to do each of the following tasks using this scale:
A— Very easy
B— Somewhat Easy
C— Somewhat Difficult
D— Very Difficult
E— N/A (I did not attempt this action)

	Easy Difficult				*N/A*
Uploading code from within jGrasp	A	B	C	D	E
Uploading code directly on the website	A	B	C	D	E
Evaluating syntax results (dealing with code style, formatting, indention, etc.)	A	B	C	D	E
Evaluating test case results (dealing with function correctness, coverage, etc.)	A	B	C	D	E

3) Do you have any comments about the WebCat software or recommendations on how to improve it?

Appendix E

Sample Introductory Training Script

1. **Welcome**
   Thanks again for participating in this study, and for meeting me today. This is an introduction to an automated software-testing tool that you will be using on your final three projects of the semester. I hope you've brought your laptops so we can get you set up. This entire process should only take about 20 minutes.

2. **Introduction: What is Web-CAT?**
   Web-CAT is a tool developed by the computer science department at Virginia Tech. It was designed to be an automated grading environment that students could submit their code to and receive an instantaneous grade. However, for this class we will only be using this tool for automated software testing. This means that you will be able to submit your code to the tool electronically and instantly receive feedback about your software's performance. You will still, however, be required to submit your finished work using the method specified by your instructor.

3. **Demonstrate Web-CAT**
   a. Test with an error
      i. Load jGrasp with sample project containing an error
      ii. Submit code to Web-CAT
      iii. Go to website and view results
   b. Fix the error and re-test
      i. Fix error in code and re-submit
      ii. Go to website and view results (note that problem is fixed)
   c. Student submitted test cases
      i. Submit the same code with an extra student-written test
      ii. Go to website and view results (note that new bug is found)

4. **What's going on here?**
   When I submit my code to Web-CAT, jGrasp is sending a copy of my code to a secure server hosted here in the computer science department. The server compiles your code and runs it against a series of test cases set up by your instructor. The server will then show you a percentage score based on these tests, along with suggestions or hints that will help you know what part of your code to improve. I will be providing you with a handout that you can use to set up your copy of jGrasp to work with this tool.

   You are also able to optionally submit your own test cases using the JUnit testing framework. You will receive a handout with more information on how to set up these test cases as well.

5. **Any questions?**

6. **Distribute handouts and set up laptops**

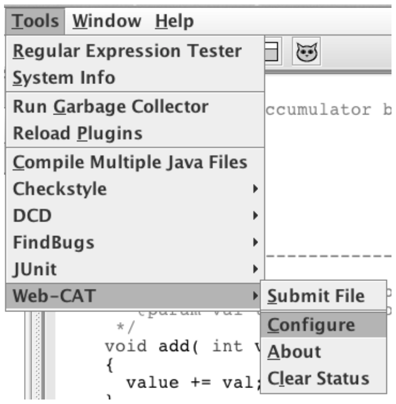
Appendix F

Web-CAT Setup Handout

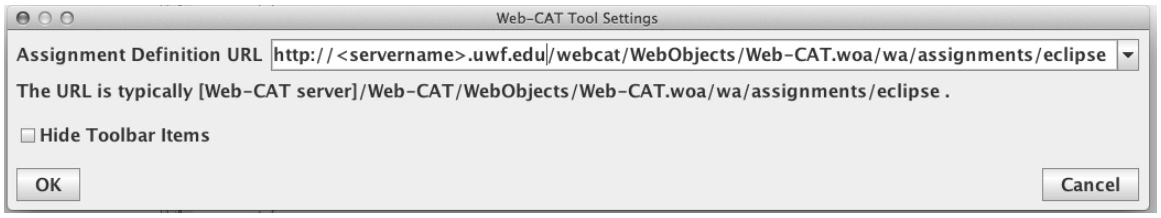
Setting up and using Web-CAT within jGrasp

Web-CAT is an automated software-testing tool. Support for Web-CAT is built into jGrasp version 1.8.8 or later. You may want to check your version if you have an older copy, but if you have just downloaded jGrasp and installed it this semester you should be fine.

1. To configure jGrasp to work with Web-CAT, start by going to Tools > Web-CAT > Configure.



2. We now need to tell jGrasp the URL it needs to use to talk to Web-CAT. Enter the following URL into the field shown below.



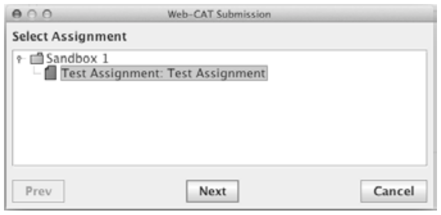
Assignment Definition URL http://<servername>.uwf.edu/webcat/WebObjects/Web-CAT.woa/wa/assignments/eclipse

The URL is typically [Web-CAT server]/Web-CAT/WebObjects/Web-CAT.woa/wa/assignments/eclipse .

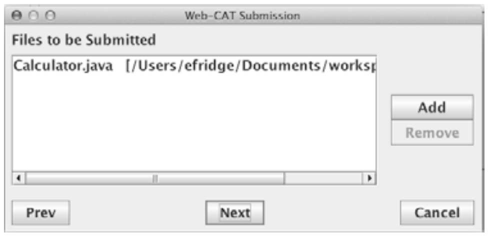
☐ Hide Toolbar Items

3. Once the path has been entered, your configuration should be complete. You can test this by opening a .java file and going to Tools > Web-CAT > Submit File.

4. You will then be asked to select the assignment that you are submitting for.



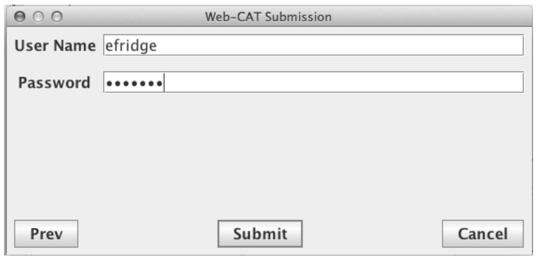
5. The next dialog box asks which files you will be submitting. You will need to submit all files needed to compile and run your project, along with any optional test cases that you have written.



Files to be Submitted

Calculator.java [/Users/efridge/Documents/worksp

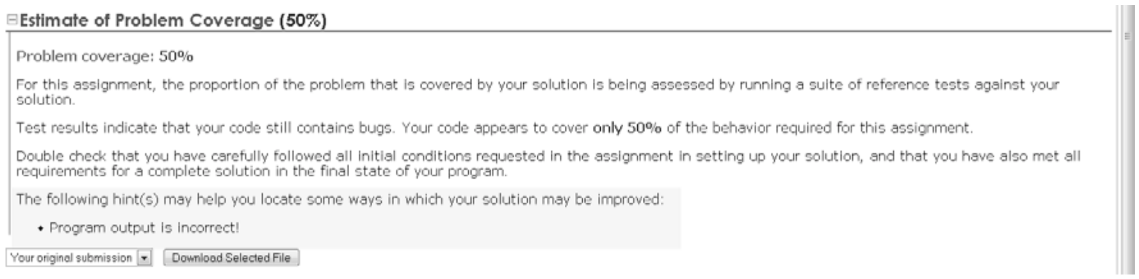
6. You will then be asked to provide your login to the Web-CAT system.



User Name: efridge

Password: •••••••

7. Your results will be displayed in your web browser once the system has finished testing your work. Please pay special attention to the "Estimate of Problem Coverage" section. This will show the percentage of test cases that passed, as well as any hints that may help you improve your work.



Estimate of Problem Coverage (50%)

Problem coverage: 50%

For this assignment, the proportion of the problem that is covered by your solution is being assessed by running a suite of reference tests against your solution.

Test results indicate that your code still contains bugs. Your code appears to cover **only 50%** of the behavior required for this assignment.

Double check that you have carefully followed all initial conditions requested in the assignment in setting up your solution, and that you have also met all requirements for a complete solution in the final state of your program.

The following hint(s) may help you locate some ways in which your solution may be improved:

- Program output is incorrect!