Student Learning and Use of Tools in an Undergraduate Software Testing Class

Dr. Peter J Clarke, Florida International Univeristy

Peter J. Clarke received his BSc. degree in Computer Science and Mathematics from the University of the West Indies (Cave Hill) in 1987, MS degree from SUNY Binghamton University in 1996 and PhD in Computer Science from Clemson University in 2003. His research interests are in the areas of software testing, software metrics, model-driven software development, domain-specific modeling languages, and computer science education. He is currently an associate professor in the School of Computing and Information Sciences at Florida International University. He is a member of the ACM (SIGSOFT, SIGCSE, and SIGAPP); IEEE Computer Society; and a member of the Association for Software Testing (AST).

Dr. Debra Lee Davis, Florida International University
Raymond Chang Lau, Florida International University
Dr. Tariq M. King, Ultimate Software Group, Inc.

Dr. Tariq M. King received his Ph.D in Computer Science from Florida International University (FIU) in 2009, and his Masters in Computer Science in 2007 from the same institution. He also holds a Bachelors in Computer Science from the Florida Institute of Technology. Dr. King is currently the Lead Software Test Architect at Ultimate Software, and an Adjunct Professor in Computer Science at FIU. He has developed and taught several software engineering courses at the graduate and undergraduate levels in academia, and has also trained software professionals in industry. His research areas of interest include software testing, autonomic and cloud computing, model-driven software engineering, and computer science education. He is a member of the ACM and IEEE Computer Society.

Observations on Student Use of Tools in an Undergraduate Testing Class

Abstract

Although practical training in software testing tools and methodologies are vital for ensuring software quality in industry, academic course curricula do not appear to be providing students with enough hands-on experience in software testing. Furthermore, there are few research studies that discuss how different pedagogical approaches to such training are helping students to improve their testing skills.

In this paper we describe how testing tools are introduced and used in an undergraduate testing course at Florida International University. As part of a semester-long course project, students access self-study tutorials on black-box and white-box testing tools via WReSTT – *a Web-Based Repository of Software Testing Tutorials*. We have captured the results of their experience in a case study. Our findings suggest that code coverage tools and techniques are an effective motivator for students to improve the quality of their test cases.

1 Introduction

As software becomes more ubiquitous, the need to improve its quality is becoming more important. Software bugs continue to plague many industries ranging from business to the military,[24] and are costing companies in the global economy in excess of \$300 Billion per year.[6] As a result of the high cost of these bugs some companies are now requiring their developers to have some form of training in software testing. This is mainly due to the fact that software testing continues to be one of the most widely used and effective means of software validation.

Although a larger number of academic institutions are expanding their curriculum to include additional software engineering courses, more needs to be done in exposing students to software testing and the use of software testing tools.[2] During the past decade there has been a noticeable improvement in the number and quality of software testing tools that have become available for use by students in academic institutions.[23] Some of these tools are so common that they are now being integrated into IDEs used to develop software, e.g., JUnit.[11] The easy access to testing tools provides interesting pedagogical research questions that can be asked. How are these tools used in the classroom? How is the easy access to tools improving the students' testing skills?

In this paper we attempt to answer the aforementioned questions by describing our experiences of using testing tools in the undergraduate testing class at Florida International University. We provide an overview of the pedagogy used in the testing class, the structure of the class and the class project, and more importantly, describe a study that shows how a cross-section of testing tools are used in the class project. It is worth noting that little or no time is spent during the class describing how to use the testing tools. Students are required to access the various tutorials in *WReSTT - Web-Based Repository of Software Testing Tutorials* that describe how to use functional testing tools and code coverage tools.[9]

In the study reported in this paper we attempt to answer the following questions: (1) Does the use of code coverage tools motivate students to improve their test suites during testing? (2) Do the results generated by the code coverage tools support the subsumes relation between branch coverage and statement coverage, i.e., does branch coverage subsume statement coverage? (3) Do students find WReSTT a useful learning resource for testing techniques and tools? (4) Do students find the features in WReSTT support collaborative learning?

To the best of our knowledge, this is one of the few studies that directly address the first two questions previously stated. Our results show that when students use coverage tools during testing they are motivated to improve their initial test suites and seek to obtain better coverage results. In our study, several code coverage tools were used which exposed students to the fact that for the same test suite, different tools can give different coverage percentages for the same criterion. The study also confirms one of the theoretical concepts taught in many testing classes, that is, branch coverage subsumes statement coverage. In many test classes the subsumes relationship is only shown with trivial examples, however in our study, students are actually able to confirm that branch coverage subsumes statement coverage. Finally, the results of the study also confirm students' perception of WReSTT as they relate to the usefulness of the tutorials and collaborative learning features in WReSTT.[8]

The remainder of the paper is organized as follows. Section 2 briefly introduces testing concepts and tools. Section 3 describes the structure of the testing course and how WReSTT is used in the course. Section 4 describes the study including the methods, results and analysis. Section 5 presents the related work and we conclude in Section 6.

2 Background

In this section we provide an overview of testing concepts and a brief description of the testing tools used in the class described in Section 3.

2.1 Testing Concepts

The Software Engineering Body of Knowledge (SWEBOK)[5] defines software testing as the *dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior*. Implicit in the definition is the execution of a program under specified conditions, observing and/or recording the results of the program execution, and making an evaluation of some aspect of the program.

There are many facets to testing including the testing levels (unit, integration, system), the view of the component under test (white-box, black-box, grey-box), the coverage criteria used to determine the effectiveness of testing (function, control-flow and data-flow), the objectives of testing (regression, acceptance, alpha), among others.[3–5, 18, 19]

In this paper we will focus on testing levels, view of the component to be tested (white-box and black-box), and the coverage criteria (control-flow - statement and branch coverage). Black-box testing techniques generate test cases based on the specification of the component, while white-box testing techniques guide the generation of test cases based on properties of the implementation, such as statement coverage and branch coverage.

The test components are then integrated into more complex components e.g., subsystems, focusing on faults discovered during the integration process. Finally the system is produced after integrating the main subsystems, and it is then tested as a whole using the requirements of the system. Since a finite set of test cases is used during testing, applying code coverage criteria during testing provides some confidence in the effectiveness of the test set.

2.2 Testing Tools

There is a plethora of software testing tools that have become available in the last decade.[23] Testing tools can play a key role in teaching software testing, particularly to students who have never been formally exposed to testing. Tools are categorized based on several criteria the most common being the testing levels, the view of the component under test (CUT) and the coverage criteria (statement, branch, basis path, among others).

In this paper we focus on the tools students used in the class projects described in Section 3.3, which include:

- *JUnit* - a free unit testing framework for the Java programming language.[11]
- *Rational Functional Tester (RFT)* - a commercial automated functional and regression testing tool.[16]
- *EclEmma* - a free Java code coverage tool for web and desktop applications.[15] Coverage metrics include instruction, statement, branch, among others.
- *eCobertura* - is a free Java code coverage reporting tool for Eclipse.[14] Coverage metrics include statement and branch.
- *CodeCover* - is a free white-box testing tool for Eclipse developed in 2007 at the University of Stuttgart.[20] Coverage metrics include statement, branch, loop, among others.

3 Undergraduate Testing Course

In this section we introduce the *CEN4072 Fundamentals of Software Testing* course that is taught at Florida International University (FIU). In addition, we briefly introduce the *Web-Based Repository of Software Testing Tutorials* (WReSTT)[8] that students use to supplement the instruction on software testing tools. This course has become very popular at FIU since several of the local companies recruit students after they have taken the course as either full-time employees or interns.

3.1 Pedagogical Approach

In the testing course we use a combination of *collaborative learning*[22] and *problem-based learning*[21] to get students involved in learning testing concepts and how to use testing tools. Collaborative learning is mainly used in the team project, described in a subsequent section, where students are placed into teams and are required to test the implementation of a software product. Collaborative learning is also encouraged at the class level where students may receive extra credit if sharing a resource helps the students in the class to better understand testing concepts, and/or learn how to better use testing tools.

By employing collaborative learning in this way students are forced to get involved in the learning process since they are required to complete some aspect of the project. The individual's work is then validated by the team, the entire class and the instructor through a series of presentations and deliverables. Team work also provides the students with the opportunity to work together, although they may have different views, abilities and personalities. Teams are randomly selected which usually results in teams with teammates who have never worked together. These characteristics of the team force members to assist each other, learn how to resolve their differences and build consensus, all key tenets of collaborative learning. Sometimes the instructor needs to intervene when the differences in the team impede productivity.

A key aspect of our approach in the testing class is the use of a problem-based learning strategy. We encourage students to work on real-world problems, in our case, students test software projects developed by other students in the software engineering class or the senior project class. Based on the feedback received from the students in the testing class, the projects assigned to be tested exhibit many of the characteristics of real-world projects e.g., incomplete documentation, a software design that is inconsistent with good design principles (low coupling and high cohesion), and an incomplete implementation of the requirements. During the class students are encouraged to actively participate in class discussions, this is achieved by students being required to give periodic reports on the progress of their projects, and being encouraged to ask questions during project presentations. We have realized that active participation by students can be significantly improved by awarding participation points during the class.

WReSTT, described later in this section, provides students with facilities that support collaborative learning, such as (1) a class-wide electronic forum where students can rate learning resources provided by other students; (2) the ability for a team to earn points based on the participation in various tasks, e.g., points awarded to a team for collectively completing online quizzes; and (3) social networking features such as, (a) activity streams showing other students completing various tasks in real-time, e.g., reading a tutorial or completing a quiz, and (b) real-time updates to their individual profiles showing the points they have earned on WReSTT after completing a task.

3.2 Course Structure

The CEN4072 course topics as stated in the catalog description are: test plan creation, test case generation, program inspections, black-box testing, white-box testing, GUI testing, and the use of testing tools. The prerequisite for CEN4072 is data structures. The students in the course are evaluated based on three exams, a team-based project, and class participation. Details on the course project are provided in the next subsection.

The material presented to students during the course is centered around black-box and white-box testing techniques.[3,18] The black-box testing techniques presented include: random, equivalence partition, boundary-value analysis and state-based testing. *Random* testing is a technique that selects random independent inputs from the domain. *Equivalence partition* testing is a technique that selects inputs from each equivalent class in the domain. *Boundary-value analysis* testing is a technique that selects values on the boundaries of the equivalent classes. *State-based* testing is a formal technique that selects inputs to traverse the state machine that represents the CUT.

The white-box testing techniques presented include: control flow coverage - statement, branch, multiple condition, basis path; and data flow coverage - all definitions, all uses, and all definitions-uses. White-box testing techniques are based on the control-flow graph (CFG) which represents the flow of control in the CUT, this is usually a method or a class in the OO paradigm. The coverage criteria are: *Statement* - percentage of all the nodes covered in the CFG or of all the statements covered in a CUT. *Branch* - percentage of all the edges covered in the CFG or of the paths covered from all conditional statements in the CUT. Additional details for the other testing techniques can be found in the literature.[3,5,18] In the CEN4072 course all testing techniques are presented through the lens of the object-oriented (OO) paradigm.[4,19]

3.3 Course Project

The CEN4072 team-based project is a key component in the course, particularly from the point of view of the students understanding the difficulty in testing a piece of software written by other developers. The students are expected to gain experience in terms of analyzing and validating software, writing test cases, and creating test documents. The software to be tested is a project from a previous software engineering class or a senior project class. The software artifacts include: the requirements document, design document, implementation document, UML diagrams and source code.

The CEN4072 project consists of two deliverables and two in-class formal presentations. Each in-class presentation includes a slide presentation and a demonstration of the tools used to test the software. The first deliverable, *Specification-Based Test Deliverable* (SBTD), focuses on black-box testing of the software and consists of a test document, the presentation slides, and the source code including test harness and test cases. The second deliverable, *Implementation-Based Test Deliverable* (IBTD), is similar to the SBTD but focuses on white-box testing and includes the updated test harnesses and test cases. The structure of the test documents follow the basic structure of the IEEE standard for software and systems documentation.[17]

During the black-box testing phase of the project the students are required to use a tool to automate the black-box testing of the system's functionality, e.g., IBM Rational Functional Tester,[16] and a unit testing tool for unit and subsystem testing, e.g., JUnit.[11] During the white-box testing phase the students are required to use at least two white-box testing tools to check the code coverage for the test cases created during the specification-based testing phase. In the class the students use EclEmma,[15] eCobertura,[14] and CodeCover.[20] In order to make the project manageable in one semester the unit testing consists of testing two classes and one subsystem package.

3.4 WReSTT

WReSTT is a collaborative learning environment that contains tutorials on testing concepts and testing tools.[25] The structure of WReSTT and the studies conducted to determine its effectiveness in a software engineering class have already be reported in the literature.[8,9] WReSTT contains tutorials for the testing tools used by students in the class project. In addition, it contains tutorials and quizzes on testing concepts that students use to re-enforce the material covered in class.

Instructors can use WReSTT to generate reports showing which students have accessed which tutorials, and their performance on the quizzes associated with the various tutorials. The report generation feature is very important for monitoring team activities, since student teams are assigned to virtual teams in WReSTT and are required to perform tasks to obtain virtual points towards class participation. One such team activity is completing the tutorial quizzes to compete for bonus virtual points. Anecdotal evidence has shown that those teams that acquire bonus virtual points also do well on the team project. However, a more comprehensive evaluation is needed before drawing any conclusions.

One of the initial issues with WReSTT was keeping the tool tutorials up-to-date, however, in the two recent editions of CEN4072, students have created high quality tutorials in their quest for extra credit. The creation of tutorials have added another dimension to the benefits of WReSTT since it is reported in the literature that preparing tutorials on a subject improves student learning.[1]

4 Case Study

The case study reported in this section focuses on the class project, specifically how the testing tools are used in the project to improve the quality of the testing activities. We also report on aspects of a survey to determine the effect WReSTT has on the students' ability to learn various testing concepts and tools, and the impact of the collaborative learning features in WReSTT. The study reported in this paper was performed in the Fall 2012 semester. We have recently completed a follow-up study in Fall 2013 and will compare the results of both studies in a future work. The specific objectives of the Fall 2012 study are to determine if:

1. The availability and knowledge of the use of code coverage tools motivate students to improve the quality of their black-box test suites.
2. The use of the code coverage tools supports the theoretical subsumes relation between branch coverage and statement coverage, i.e., branch coverage subsumes statement coverage.
3. Students find WReSTT a useful learning resource for testing techniques and tools.
4. Students find that WReSTT supports collaborative learning.

One of the tenets of white-box testing is that there is a subsumes relation between various code coverage criteria such as statement coverage, branch coverage, multiple condition coverage, all definitions, all uses and so on. Criterion 1 subsumes criterion 2 if every test suite that satisfies criterion 1 also satisfies criterion 2.[10] Frankl et al.[10] state that test coverage criteria does not say anything about the fault detecting ability of the test suites generated based on a given criteria. However, Zhu[27] shows that under certain conditions the subsumes relation does guarantee a better fault detecting ability. Zhu further states that to improve fault detection in practical situations a stricter adequacy criterion in the subsume hierarchy should be used where possible.[27]

4.1 Methods

Sample: The students that participated in the study were from the Fall 2012 CEN4072 Fundamentals of Software Testing class at Florida International University (FIU). The class contained 35 students which were assigned to 6 project teams.

Data Collection: Data for the study was collected using: a review of the artifacts for the two project deliverables (SBTD and IBTD) - documents, presentation slides, and source code; observation of the in-class team presentations; and a survey instrument consisting of 30 questions divided into 5 sections. The reason for observing the in-class presentations was to determine if the teams used the tools to fully automate the running of the test cases. The survey used to evaluate the students interactions with WReSTT is available on the Project Documentation page of WReSTT[a]. Note that we only used questions 16 - 26 of the survey, those that focus on the tutorials related to testing concepts and tools (16 -20), and the collaborative learning features in WReSTT (21 - 26).

All project teams were assigned the same software engineering project, titled "PantherLot Interactive" from the summer 2011 software engineering class. PantherLot Interactive is a garage parking system that is expected to reduce the time for the members of the FIU community to find an available parking spot. The parking system uses scanners and sensors to determine if there are any free parking spots before drivers enter a parking lot. The system serves four types of users, students, faculty, administrators, and guests. All hardware specific devices such as scanners and sensors were simulated by software modules. PantherLot Interactive uses a three-tier architecture consisting of 5 packages and 24 classes.

Design: The software to be tested was made available to the project teams during the second class of the semester. This was to allow the students to get the software running. The SBTD and IBTD were due during weeks 8 and 15 of the semester, respectively. At the beginning of the semester the students were informed of this study and how the documents would be reviewed. Students were enrolled in WReSTT during week 2 of the semester and were provided access to the tutorials at that time. The WReSTT survey was administered during the last week of the semester.

During black-box testing for the first deliverable, the students were instructed to create test cases based on the use cases of the system as follows. For each use case there should be at least 2 sunny day scenario test cases, and 1 rainy day scenario test case. By sunny day scenario we mean

[a]http://wrestt.cis.fiu.edu/cen/documentation

Table 1: Number of test cases created during black-box and white-box testing. System testing was done using Rational Functional Tester, subsystem and unit testing was done using JUnit.

	Initial Test Cases (Black-box Testing)			Test Cases Added (White-box Testing)		
Team	Unit	Subsys.	System	Unit	Subsys.	System
1	51	7	65	6	5	0
2	10	31	18	4	1	6
3	12	43	24	9	18	5
4	47	16	27	10	11	7
5	17	20	36	7	11	12
6	14	36	33	13	3	5

that the instance of the use case should follow the normal transaction flow, and for the rainy day scenario the transaction should include one alternative path or an exception. The students were also instructed to develop the subsystem test cases based on the system test cases. During the white-box testing phase the students were required to use the results of the code coverage tools to improve their black-box test suites. That is, based on the percentage statement and branch coverage obtained when running the original test suite, new test cases were added to improve the code coverage.

4.2 Results and Analysis

To determine if the availability and knowledge of the use of code coverage tools motivated students to improve the quality of their black-box test suites, we reviewed the project deliverables. Table 1 shows the number of test cases created by each team during black-box testing and the number of test cases added during white-box testing for different levels of testing (unit, subsystem and system). As previously mentioned the class contained 35 students assigned to 6 teams shown in Column 1 of the table. The unit and subsystem testing was done using *JUnit*[11] and the system testing was done using *IBM Rational Functional Tester (RFT)*.[16] In all cases, except one, new test cases were added during white-box testing. Note however, in that case (Team 1 system testing) the number of black-box test cases was high compared to the other teams, see Column 4 of Table 1.

Table 2 shows the percentage code coverage obtained for the black-box and white-box test suites using the various tools. The columns of the table are partitioned into three sections. Columns 1 and 2 show the team number and coverage testing tool used; Columns 3 through 8 show the statement and branch coverage for the various testing levels for the black-box test suite; and Columns 9 through 14 show the statement and branch coverage for the various testing levels for the white-box test suite. The coverage tools used were *eCobertura*,[14] *EclEmma*[15] and *CodeCover*.[20]

The first row for Team 2 shows the results obtained using eCobertura. For the black-box test suite: unit testing - 72.4% statement and 61.1% branch coverage, subsystem testing - 79.8% statement

Table 2: Code coverage for black-box and white-box testing of the class project for each team.

		Coverage (%) for Black-box Test Suite						Coverage (%) for White-box Test Suite					
		Unit		Subsystem		System		Unit		Subsystem		System	
Team	Tool	Stmt	Branch	Stmt	Branch	Stmt	Branch	Stmt	Branch	Stmt	Branch	Stmt	Branch
1	eCobertura	80.8	88.9	83.5	75.7	85.0	75.0	100.0	94.4	86.7	81.1	85.0	78.0
	EclEmma	37.1	-	80.8	-	-	-	79.9	-	84.1	-	-	-
	CodeCover	44.2	30.9	81.5	67.0	-	-	90.7	78.4	86.0	73.6	-	-
2	eCobertura	72.4	61.1	79.8	76.6	43.0	39.0	98.9	92.5	87.0	81.5	77.0	73.0
	EclEmma	35.5	23.9	79.5	76.6	-	-	81.9	67.5	86.9	81.5	-	-
	CodeCover	37.2	13.2	73.6	58.3	-	-	92.2	67.6	81.3	60.7	-	-
3	eCobertura	50.0	50.0	70.5	61.9	77.0	71.0	100	100	98.6	96.1	89.0	85.0
	EclEmma	49.4	-	71.1	-	-	-	100	-	98.9	-	-	-
	CodeCover	-	-	-	-	-	-	-	-	-	-	-	-
4	eCobertura	69.5	52.7	69.2	58.9	79.0	67.0	87.1	86.1	87.3	83.9	89.0	83.0
	EclEmma	61.4	-	68.0	-	-	-	78.0	-	84.7	-	-	-
	CodeCover	67.9	38.7	61.6	52.1	-	-	89.4	72.2	86.3	76.6	-	-
5	eCobertura	71.0	57.1	42.4	33.3	83.0	78.0	97.9	100	90.0	83.3	84.0	80.0
	EclEmma	44.9	26.2	36.5	91.9	-	-	81.1	66.7	85.3	80.9	-	-
	CodeCover	-	-	39.3	28.4	-	-	-	-	91.3	81.1	-	-
6	eCobertura	56.0	48.0	43.9	36.2	72.0	78.0	97.8	100	90.6	79.7	73.0	81.0
	EclEmma	29.9	22.0	42.4	29.4	-	-	76.3	68.0	56.0	53.8	-	-
	CodeCover	57.3	-	31.0	-	75.0	-	87.0	-	71.3	-	77.0	-
Average		54.0	42.7	62.0	57.4	73.4	68.5	89.9	82.8	85.4	78.0	82.0	80.0
Std Dev.		15.4	21.2	18.4	20.5	14.1	15.2	8.6	14.2	9.8	10.6	6.4	4.2

and 76.6% branch coverage, system testing - 43.0% statement and 39.0% branch coverage. After white-box testing the updated test suite resulted in: unit testing - 98.9% statement and 92.5% branch coverage, subsystem testing - 87.0% statement and 81.5% branch coverage, system testing - 77.0% statement and 73.0% branch coverage.

We used descriptive statistics to summarize the results shown in this table in the last two rows of the table. The second to last and last rows of Table 2 show the average and standard deviation of the values in the preceding rows, respectively. For example the average statement coverage obtained during black-box testing is 54.0% and the standard deviation is 15.4. Note the standard deviation is high for black-box testing since there is no black-box testing technique that guides you on how to select a minimal subset of test cases from a usually infinite input space. The standard deviation is smaller for the white-box testing since code coverage helps to guide the selection of test cases.

Figure 1 shows the average percentage increase in the number of test cases, statement coverage, and branch coverage for each testing level between the black-box testing phase and the white-box testing phase. This figure shows the average for all teams. The leftmost group of bars represent unit testing, the group in the middle subsystem testing, and the the rightmost group system testing. Each group consists of three bars, increase percentage of test cases on the left, increase percentage of statement coverage in the middle, and and increased percentage of branch coverage on the right.

Table 3 shows the students' perceptions of the usefulness of the tutorials available in WReSTT.

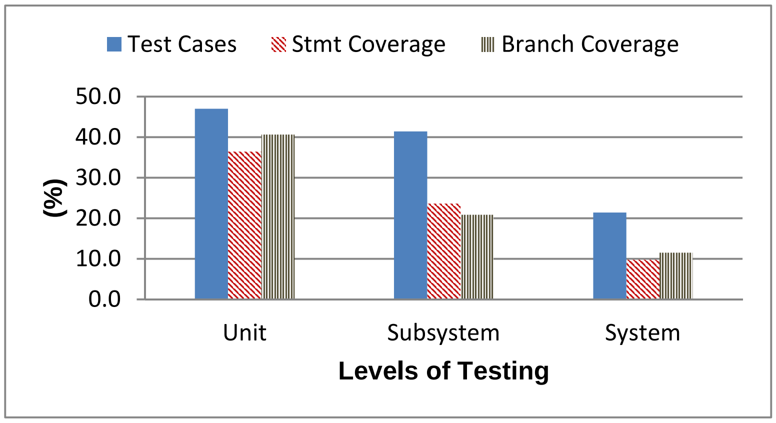

Figure 1: Average percentage increase in the number of test cases, statement coverage, and branch coverage for each testing level for all the teams.

These results were positive for Questions (Q.) 16 - 19 with all the scores above 3.24. These questions focused on the effectiveness of WReSTT in helping students to understand testing concepts and how to use the testing tools. Q.20 (M = 2.97, SD = 1.16), the lowest value in the table that focuses directly on the usefulness of WReSTT, shows the students felt that the number of tutorials were somewhat inadequate.

Table 3: Students' mean scores (standard deviations) in Section 3 of the survey measuring perception of the usefulness of testing tutorials in WReSTT.

Q.	Testing Related Questions	N	M (SD)
16	The tutorials in WReSTT helped me to better understand testing concepts.	33	3.97 (0.85)
17	The tutorials in WReSTT helped me to better understand how to use unit testing tools.	33	4.06 (0.86)
18	The tutorials in WReSTT helped me to better understand how to use code coverage testing tools.	32	3.45 (1.35)
19	The tutorials in WReSTT helped me to better understand how to use functional testing tools.	32	3.24 (1.39)
20	The number of tutorials in WReSTT is adequate.	33	2.97 (1.16)

Table 4 shows the students' perceptions of the usefulness of the collaborative learning features in WReSTT. The table shows the results for the Questions (Q.) 22 - 26 and there were all very positive with scores above 3.79 out of 5. These questions focused on how WReSTT motivated students to collaborate with team members to complete the various tasks in WReSTT. As previously stated students were awarded virtual points based on their participation in team activities, as well as their individual participation. It is very interesting to note that Q. 23 (M = 4.33, SD = 1.36) was the highest value in the table, which showed that the assignment of virtual points encourage team members to visit WReSTT and complete the assigned tasks.

Table 4: Students' mean scores (standard deviations) in Section 4 of the survey measuring usefulness of the collaborative features in WReSTT.

Q.	Collaborative Learning Related Questions	N	M (SD)
22	The use of virtual points in WReSTT encouraged me to visit the website and complete the tasks.	32	4.27 (1.44)
23	The use of virtual points in WReSTT encouraged my team to visit the website and complete the tasks.	32	4.33 (1.36)
24	The event stream showing the activities of the other members in the class encouraged me to complete my tasks in WReSTT.	33	3.85 (1.28)
25	The event stream showing the activities of the other members in the class encouraged my team to complete my tasks in WReSTT.	32	3.79 (1.32)
26	Our team devised a plan to get the maximum number of points in WReSTT.	33	4.03 (1.02)

4.3 Discussion

Based on the results presented in the previous subsection we can infer the following with respect to the objectives of the study.

Objective 1: The results show that the availability and knowledge of the use of code coverage tools motivated students to improve the quality of their black-box test suites. The results in Tables 1 and 2, and summarized in Figure 1, show that for all teams new test cases were added during white-box testing in order to achieve a higher statement and branch coverage. The only exception was the system level testing for Team 1 who was at 85% statement coverage and assumed this was good enough. In some cases teams attempted to obtain 100% statement coverage, see the white-box testing at the unit level for Team 1 and 3 in Table 2. One positive side effect of the white-box testing activity was students realized that it was sometimes infeasible or difficult to achieve 100% code coverage for a given criterion.

Objective 2: One of the tenets of white-box testing is the subsumes relation for code coverage criteria, previously described. Many of the testing tools only consistently provide statement and branch coverage.[26] From the results shown in Table 2, second to last row, labeled Average, the value in the branch coverage column was consistently lower than the values for statement coverage. The results of the study support the claim that branch coverage subsumes statement coverage. In future studies we plan to investigate other subsumes relations, e.g., multiple condition coverage (MCC) subsumes branch coverage. One limitation of extending the study is that most freely available tools only provide statement and branch coverage, which is two of the less efficient white-box testing techniques as reflected in the subsume hierarchy.

Objective 3: The results in Tables 3 show that students do find WReSTT a useful resource for learning software testing techniques and how to use testing tools. Students found WReSTT particularly helpful with understanding testing concepts and how to use unit testing tools. This part of the study shows that WReSTT is useful not only for software engineering classes, as reported by Clarke et al.,[8] but also for software testing classes, however, to a lesser extent. The main difference is that in the software engineering class the students found that the number of tutorials were more adequate than in the testing class. This is one area of WReSTT that is currently being remedied by soliciting tutorials from the testing educators, as well as encouraging students to create new tutorials.

Objective 4: The results in Table 4 show that students perceived WReSTT as supportive of collaborative learning since it encourages them to work collectively within their teams to complete tasks, and compete against other teams for virtual points. Students thought that the use of virtual points was a big motivating factor for them to work in teams. They also felt positive about the event streaming feature showing how other classmates were doing, but ranked it the lowest of the collaborative learning features in WReSTT. As compared to a previous study by Clarke et al.,[8] students in the testing class were not as enthusiastic about the collaborative learning features in WReSTT, as the students in the software engineering class.

Threats to Validity. The main threat to validity is the lack of completeness of the results shown in Table 2. Clearly not all teams were knowledgeable on how to use all the features of the code coverage testing tools. This is evident when we compare the results obtained by Team 2 and the other teams. Team 2 was the only team able to consistently obtain statement and branch coverage using all coverage tools when performing unit and subsystem testing. Note however there was a problem for all teams obtaining code coverage results when performing system testing using Rational Functional Tester (RTF).[16] Students were not able to instrument the code then used the capture play back mechanism in RTF when running test cases.

During the observation of the in-class demonstrations and review of the project documents, it was clear that some teams could not interpret the results presented in the reports generated by the code coverage tools. For example, there were some inconsistencies with the screen-shots of the tool reports that were available in the documents and the in-class presentation. In some cases the teams reported the results for system coverage when they were actually doing subsystem testing of a package. We plan to repeat the study in a future class and remedy these threats. We also feel that a follow up study is needed before any general conclusions can be made about the impact of using testing tools in an undergraduate testing class.

5 Related Work

Carver et al.[7] conducted an empirical study to evaluate the testing ability of senior-level CS students in a software engineering course. The study consisted of two steps: (1) the manual creation of a test suite, and (2) the use of a code coverage tool, CodeCover,[20] to improve the test suite. The students were required to test two programs of approximately 200 SLOC each. It was concluded that students had trouble obtaining 100% code coverage unless a testing tool was used. Our work is similar to the work by Carver et al., except we are performing the study on a software testing class and are using a larger software project. Our project is so large that it is not expected that students identify a test suite to get a high percentage of coverage unless tools are used.

Harrison[13] describes the structure of the testing course at his university where testing is taught using two viewpoints, that of the developer and the tester. The course uses the project format where students are required to test the program on which they are the developers, then test the program from another team where they act solely as testers. The results show that students learn the intended skills and the workload is appropriate. One positive experience for the students is that they are exposed to student groups using different testing techniques. Unlike the work presented by Harrison, students mainly use the point of view of a tester. However, they are required to use different testing techniques, including both blackbox and white box testing techniques.

Garousi[12] describes an approach of incorporating real-world industrial testing projects into a graduate testing course. A description of the course is presented in the paper and the testing projects are described. The lessons learned from this approach include: the need to proactively monitor student progress, need to be realistic about the workload associated with such projects, and the need to ensure students are not only focused on learning how to use the tools. In our work we provide students the next best thing to working on a real-world project, testing a software engineering project from a previous class. The students usually complain how difficult it is to run the software and understand the code.

Clarke et al.[8] present an empirical study to evaluate a non-intrusive approach of integrating testing techniques into software engineering courses using WReSTT. The results of that study showed that the use of WReSTT can: improve the students' understanding and use of software testing techniques and tools; it is a useful learning resource for software testing; and it supports collaborative learning. In this paper we repeated the aspect of the study that is related to the usefulness of WReSTT as a learning resource in a software testing class. The results of our study show that WReSTT is useful both as a learning resource by providing tutorials, and as a collaborative learning environment by supporting team activities. However, students in the software engineering class found WReSTT to be more useful that in the software testing class.

6 Conclusions

In this paper we presented our experiences of using tools in an undergraduate software testing class at Florida International University. A description of the pedagogical approach, software testing course structure, and the course project were presented. In addition, we described how both black-box and white-box testing tools were used during the class to test implementation of the project. We conducted a case study during the class and it was determined that: (a) the availability and knowledge of the use of code coverage tools motivated students to improve the quality of their black-box test suites; (b) the subsume relation between statement coverage and branch coverage holds; and (3) WReSTT - Web-Based Repository of Software Testing Tutorials, is a useful resource for learning software testing. We have currently collected additional data for a follow up study and will perform a comparative study to determine if the results presented in this paper are conclusive.

Acknowledgments

This work was supported in part by the National Science Foundation under grants DUE-0736833, DUE-1225742 (FIU) and DUE-0736771 (FAMU). We would like to thank the students in the CEN4072 Fundamentals of Software Testing class that participated in the study. We also appreciate the insightful comments provided by the reviewers on how to improve the paper.

References

[1] AAAS and NSF. Transforming undergraduate education in stem: Making and measuring impacts. 2013 tues principal investigators (pis) conference, January 2013. `http://ccliconference.org/files/2013/01/2013-Tues-Conference-Program.pdf`.

[2] ACM/IEEE-CS Interim Review Task Force. Computer Science Curriculum 2008: An Interim Revision of CS 2001., December 2008. `http://www.acm.org/education/curricula/ComputerScience2008.pdf`.

[3] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[4] R. V. Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[5] P. Bourque and R. Dupuis. *Guide to the Software Engineering Body of Knowledge 2004 Version*. IEEE Computer Society, Los Alamitos, California, 2004.

[6] G. Carver, L. Jeng, P. Cheak, T. Britton, and T. Katzenellenbogen. Cambridge university study states software bugs cost economy $312 billion per year, January 2013. `http://www.prweb.com/releases/2013/1/prweb10298185.htm`.

[7] J. Carver and N. Kraft. Evaluating the testing ability of senior-level computer science students. In *24th IEEE-CS Conference on CSEET, 2011*, pages 169–178, 2011.

[8] P. J. Clarke, J. Pava, D. Davis, and T. M. King. Using WReSTT in SE courses: An empirical study. In *Proceedings of the 43rd SIGCSE Conference*, pages 307–312, New York, NY, USA, 2012. ACM.

[9] P. J. Clarke, J. Pava, Y. Wu, and T. M. King. Collaborative web-based learning of testing tools in SE courses. In *Proceedings of the 42nd SIGCSE Conference*, pages 147–152, New York, NY, USA, 2011. ACM.

[10] P. Frankl and E. Weyuker. A formal analysis of the fault-detecting ability of testing methods. *IEEE Transactions on Software Engineering*, 19(3):202–213, Mar 1993.

[11] E. Gamma and K. Beck. JUnit, May 2012. `http://www.junit.org/`.

[12] V. Garousi. Incorporating real-world industrial testing projects in software testing courses: Opportunities, challenges, and lessons learned. In *24th IEEE-CS Conference on CSEET, 2011*, pages 396–400, 2011.

[13] N. B. Harrison. Teaching software testing from two viewpoints. *J. Comput. Sci. Coll.*, 26(2):55–62, Dec. 2010.

[14] J. Hofer. eCobertura, August 2013. `http://ecobertura.johoop.de/`.

[15] M. R. Hoffmann, B. Janiczak, and E. Mandrikov. EclEmma, August 2013. `http://www.eclemma.org/`.

[16] IBM. Rational Functional Tester , May 2012. `http://www-01.ibm.com/software/awdtools/tester/functional/`.

[17] IEEE. IEEE Standard for Software and System Test Documentation, 2008. https://standards.ieee.org/findstds/standard/829-2008.html.

[18] A. P. Mathur. *Foundations of Software Testing*. Pearson Education India, Delhi, India, 1 edition, 2008.

[19] J. D. McGregor and D. A. Sykes. *A practical guide to testing object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[20] R. Schmidberger. CodeCover, August 2013. http://codecover.org/.

[21] C. Y. Shim, M. Choi, and J. Y. Kim. Promoting collaborative learning in software engineering by adapting the pbl strategy. In *Proceedings of WASET International Conference on Computer and Information Technology (ICCIT 2009)*, pages 1167–1170, Washington, DC, USA, 2009. IEEE Computer Society.

[22] B. L. Smith and J. T. MacGregor. What is Collaborative Learning? In A. Goodsell, M. Maher, and V. Tinto, editors, *Collaborative Learning: A Sourcebook for Higher Education.* National Center on Postsecondary Teaching, Learning, and Assessment, University Park, Pa., 1992.

[23] Wikipedia. Category:software testing tools, May 2013. http://en.wikipedia.org/wiki/Category:Software_testing_tools.

[24] Wikipedia. List of software bugs, Sept. 2013. http://en.wikipedia.org/wiki/List_of_software_bugs.

[25] WReSTT Team. WReSTT: Web-based Repository for Software Testing Tools, May 2012. http://wrestt.cis.fiu.edu/.

[26] Q. Yang, J. J. Li, and D. Weiss. A survey of coverage-based testing tools. *The Computer Journal*, 52(5):589–597, Apr. 2007.

[27] H. Zhu. A formal analysis of the subsume relation between software test adequacy criteria. *IEEE Transactions on Software Engineering*, 22(4):248–255, Apr 1996.