Integrating Testing into Software Engineering Courses Supported by a Collaborative Learning Environment

PETER J. CLARKE and DEBRA DAVIS, Florida International University
TARIQ M. KING, North Dakota State University
JAIRO PAVA, Florida International University
EDWARD L. JONES, Florida A&M University

As software becomes more ubiquitous and complex, the cost of software bugs continues to grow at a staggering rate. To remedy this situation, there needs to be major improvement in the knowledge and application of software validation techniques. Although there are several software validation techniques, software testing continues to be one of the most widely used in industry. The high demand for software engineers in the next decade has resulted in more software engineering (SE) courses being offered in academic institutions. However, due to the number of topics to be covered in SE courses, little or no attention is given to software testing, resulting in students entering industry with little or no testing experience.

We propose a minimally disruptive approach of integrating software testing into SE courses by providing students access to a collaborative learning environment containing learning materials on testing techniques and testing tools. In this article, we describe the learning environment and the studies conducted to measure the benefits accrued by students using the learning environment in the SE courses.

Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*; K.3.1 [**Computers and Education**]: Computer Uses in Education—*Collaborative learning*

General Terms: Experimentation

Additional Key Words and Phrases: Course management, code coverage, software testing, unit testing, testing tutorials

ACM Reference Format:
Peter J. Clarke, Debra Davis, Tariq M. King, Jairo Pava, and Edward L. Jones. 2014. Integrating testing into software engineering courses supported by a collaborative learning environment. ACM Trans. Comput. Educ. 14, 3, Article 18 (October 2014), 33 pages.
DOI: http://dx.doi.org/10.1145/2648787

1. INTRODUCTION

Software bugs continue to have an impact on industry, ranging from space exploration to financial businesses [Wikipedia 2012]. One business recently impacted by a software bug was the Knight Capital Group. It is estimated that as a result of a software bug, $440 million was lost in less than 1 hour due to the execution of a series of automatic

This work is supported by the National Science Foundation under grants DUE-0736833 and DUE-1225742 (FIU), and grant DUE-0736771 (FAMU). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.
Authors' addresses: P. J. Clarke, D. Davis, and J. Pava, School of Computing and Information Sciences, Florida International University; emails: clarkep@cis.fiu.edu, dledavis@cs.fiu.edu, jpava001@fiu.edu; T. M. King, Department of Computer Science, North Dakota State University; email: Tariq.King@ndsu.edu; E. L. Jones, Department of Computer and Information Sciences, Florida A&M University; email: ejones@cis.famu.edu.

DOI: http://dx.doi.org/10.1145/2648787

stock orders [CNN 2012]. As software becomes more ubiquitous and complex, and increases in size, major improvements in the knowledge and application of software validation techniques are needed [CNSS 2005; Gallaher and Kropp 2011]. One of the key requirements necessary to achieve these improvements is more and better trained professionals in the area of software validation. To create a workforce better trained in software validation techniques, curricula at academic institutions need to be updated to include instruction on the various software validation techniques, including software testing.

During the past decade, computer science (CS) and information technology (IT) educators have made an effort to integrate software testing into the CS/IT curricula at several academic institutions [Desai et al. 2009; Dvornik et al. 2011; Janzen and Saiedian 2006; Edwards 2003; Frezza 2002; Kaner et al. 2001; Jones 2000]. These approaches range from the integration of testing into CS1 and CS2 using novel approaches, such as test-driven learning (TDL) and test-driven development (TDD), to the restructuring of more advanced CS/IT courses to include a software testing component. Lethbridge et al. [2007] state that more academic institutions are offering courses in software engineering (SE), which is a good thing; however, more needs to be done to expose students to software testing techniques, testing practices, and testing tools [ACM/IEEE-CS Interim Review Task Force 2008; Astigarraga et al. 2010; Shepard et al. 2001].

In this article, we describe a minimally disruptive approach to integrating software testing into SE and upper-level programming courses. We use the term *minimally disruptive* because we do not expect instructors who use the approach to make major changes in their current teaching strategies. This provides an advantage to instructors, making it easier for them to integrate our approach into their curriculum, and thus making it more likely to be adopted and useful for a large number of institutions. This project started 5 years ago with the idea of providing students with access to an online repository of tutorials on software testing tools that would supplement the instruction on testing provided in class. Since then, the project has evolved into a cyberlearning environment that provides learning materials on both testing concepts and tools.

The initial online repository created in the project was the *Web-Based Repository of Software Testing Tools*, version 1 (WReSTT V1), which contained tutorials on software testing tools and links to other learning resources on software testing. Based on the feedback from students and instructors, as well as the results of several studies performed by the authors [Clarke et al. 2010, 2011, 2012], WReSTT V1 was transformed from a simple repository of software testing tool tutorials to a collaborative learning environment that contains a broader array of tutorials on testing concepts and testing tools (WReSTT version 2 (V2)) [WReSTT Team 2012].[1]

This article extends the prior work presented by Clarke et al. [2010, 2011, 2012], which describes (1) the design of WReSTT V1 and the preliminary studies on students' perceptions of using the resources in WResTT V1 to improve their testing skills [Clarke et al. 2010]; (2) the initial design of WReSTT V2 and the results of studies to evaluate the implementation of the collaborative learning environment [Clarke et al. 2011]; and (3) an extension of the WReSTT V2 design that include features to (a) create course templates for instructors and (b) support an instructor's ability to load class rolls, create virtual teams, and access student reports [Clarke et al. 2012]. Additional details of the prior work are presented in Section 5 of this article.

The contributions of this work are as follows: (1) provide a more comprehensive description of the high-level structural design of WReSTT V2 and (2) discuss a more extensive study to determine the impact of using the resources of WReSTT V2 to

[1]http://wrestt.cis.fiu.edu/.

improve students' testing skills in SE courses. It is important to stress that the approach reported in this article is focused on students in an SE class where there is limited time to address testing concepts in the course. The WReSTT V2 structural design is provided so that instructors can develop a similar collaborative learning environment for a specific domain of their choice, or simply use the facilities provided by WReSTT to support the instruction of testing in their classes.

The study consisted of two control and two treatment groups, and the objectives of the study focused on students' improvements in general knowledge of software testing, knowledge and use of testing techniques, knowledge and use of testing tools, usability of WReSTT V1 and V2, and the impact of the collaborative learning environment on their ability to use testing tools in their team project. The results of the study indicate that using WReSTT in SE classes can improve students' general knowledge of software testing techniques and tools. In general, students found WReSTT engaging and enjoyable to use, with students having a positive view of WReSTT V2. Finally, students who were exposed to WReSTT were more likely to use testing tools during their SE project.

The remainder of the article is structured as follows. Section 2 provides background on software testing and collaborative learning. Section 3 introduces WReSTT, focusing more on the structure and functionality of the current version of WReSTT (V2). Section 4 describes the study comparing the improvement in students' testing skills when using WReSTT. Section 5 describes the related work on the approaches used to introduce testing into programming courses and the online resources that contain learning materials on software testing. Section 6 concludes the article by describing a summary of the results of the study and future directions for WReSTT.

2. BACKGROUND

In this section, a brief background of software testing is provided to introduce some of the key concepts related to the work presented in this article. These concepts include the different levels of testing, the view of the component to be tested, and the coverage criteria used during testing. In addition, we introduce collaborative learning, problem-based learning (PBL), and gamification—three key learning strategies used in designing WReSTT.

2.1. Software Testing in an SE Course

There are several definitions for software testing; however, the one that we use is from Software Engineering Body of Knowledge (SWEBOK) [Bourque and Dupuis 2004]. The definition states that software testing is 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 based on various characteristics of the implementation. There are many facets to testing, including the testing levels (unit, integration, system), the objectives of testing (regression, acceptance, alpha), testing focusing on nonfunctional requirements (usability, security, performance), the view of the component to be tested (white box, black box, grey box), and the coverage criteria used to determine the effectiveness of testing (function, control flow, and dataflow) [Ammann and Offutt 2008; Binder 1999; Bourque and Dupuis 2004; Mathur 2008; Myers 2004].

As the size and complexity of software has grown, the use of tools to support the testing process has become essential. Perry [2006] states in his book on software testing that the use of tools should always be mandatory. Using tools in the classroom helps students understand and reinforce many software testing concepts, as well as improve

their practical testing skills. For example, using JUnit [Gamma and Beck 2012] helps students improve their practical skills for setting up/tearing down a test environment, writing stubs and drivers, and using the assert statement to compare the actual and expected results for a test case. The use of coverage tools also helps students improve their test case writing skills because they always strive to get better code coverage with the new test suite.

Teaching software testing in an SE course is usually a challenge for the following reasons: (1) there are usually too many other topics to cover in an SE course, (2) many SE instructors are not familiar with software testing techniques, and (3) students and instructors are not familiar with the tools available to support software testing. A quick scan of several SE textbooks [Bruegge and Dutoit 2009; Pfleeger and Atlee 2009; Sommerville 2004] will reveal the large number of topics to be covered in an introductory SE course. In addition, many of the software development models presented in these textbooks usually cover software engineering topics such as requirements elicitation and analysis, systems and detailed design, and implementation, which are considered to be more glamorous than testing [Cowling 2012].

2.2. Collaborative Learning

Smith and MacGregor [1992] state that *collaborative learning* represents several educational strategies that involve intellectual effort by students or students and teachers working together. These educational strategies vary widely but focus mainly on students' exploration or application of course material, not on the presentation of the material by the teacher. Smith and MacGregor also state that collaborative learning promotes several goals of education, including *involvement*, in which students are more involved in the learning process by interacting more with other students and teachers; *cooperation and teamwork*, in which students working together will be confronted with different views and will therefore need to resolve these differences and build consensus in their teams; and *civic responsibility*, which encourages students to participate in shaping their ideas and values.

Teaching SE lends itself to collaborative learning due mainly to the project, which is usually an inherent part of the course. Several studies have shown how collaborative learning and other pedagogical strategies can be integrated into SE courses. Shim et al. [2009] describe an approach to promote collaborative learning in an SE course using a PBL strategy. The authors' approach is based on integrating the following PBL characteristics into the software process, particularly in the development of the course project. These PBL characteristics include (1) the use of real-world problems, (2) encouragement of students' active participation, (3) integration of diverse view points, (4) encouragement of self-oriented learning, (5) encouragement of collaboration, and (6) enhancement of education quality.

One other learning approach that we incorporate in WReSTT is *gamification*—that is, using game design elements and game mechanics to increase user experience and engagement with a system [Domínguez et al. 2013]. Malone [1980] did some of the earlier work on emphasizing how attributes of games can be used in an educational context. More recent work by Li et al. [2013] describes how gamification is used to engage CS students in an online social network–based collaborative learning environment, *PeerSpace*, to enhance CS student learning in several CS courses. The game mechanics used in PeerSpace include participant points, participation levels, and leader boards. We use these attributes of gamification in WReSTT to improve student engagement.

3. USING WReSTT TO IMPROVE SOFTWARE TESTING PEDAGOGY IN SE COURSES

In this section, we describe the evolution of *Web-Based Repository of Software Testing Tutorials*(WReSTT V2), its current design, and how it may be used in the classroom.

WReSTT was initially designed to support the pedagogical needs of students and instructors in programming and SE courses by providing access to a comprehensive and up-to-date set of tutorials on tools that aid software testing. Based on the feedback from students and instructors, WReSTT has evolved into a collaborative learning environment with social networking features. WReSTT is available for use by instructors either by accessing it via the Internet (http://wrestt.cis.fiu.edu/) or by requesting assistance from the authors if they want to install an implementation at their institution. Note that in WReSTT V2, the last "T" in WReSTT represents "Tutorials," not "Tools" as in WReSTT V1.

3.1. WReSTT Evolution

WReSTT V1 [Clarke et al. 2010] was developed in 2008 based on a four-tier architecture and implemented using Drupal [Drupal Community 2012], a content management system. The *data store tier* contains tables of data on the users, tools, forum messages, and other feedback from users. The *application logic tier* contains an email module, a ratings module for the tools, and a module to analyze usage (e.g., access to tutorials and user page access). The *presentation tier* generates pages for and interprets data from the different categories of users. These categories are administrators, developers, instructors, and students. The *client tier* represents the interfaces displayed for the various users.

WReSTT V1 only contains learning materials on tools to support software testing that are classified by category, language, or test level. The categories of testing tools include:

—*Coverage*: Tools that compute test coverage of program source code or requirements;
—*Metrics*: Tools that perform static analysis on the program source code;
—*Plug-ins*: Tools built as add-ons to an integrated development environment (IDE);
—*Test execution*: Tools that automatically run test cases on software; and
—*Web*: Tools that replicate or simulate user actions using a Web browser.

Currently, there are tools for two languages in WReSTT V1: *CPP* and *Java*. The testing levels include *unit*, or tools that test basic software units (e.g., classes) in isolation, and *system/UI*, or tools that test the software system as a whole from its user interface. Table I shows a list of tools for which there are learning materials in WReSTT V1.

The initial selection of tools to support instruction was based on four top-level criteria: (1) tools for each of the two most popular programming languages currently used in CS and IT programs, CPP and Java [Chen et al. 2005]; (2) the results of evaluating tools based on a set of low-level criteria, such as usability, vendor/developer support, system requirements, supporting documentation, and feature support [Crowther and Clarke 2005]; (3) tools for different languages belonging to same family of testing tools, such as the xUnit family: CppUnit and JUnit; and (4) tools that use different testing approaches, such as unit testing, code coverage, and systems testing. Clarke et al. [2010] provide additional details on WReSTT V1.

During fall 2009, a pilot study was conducted on the undergraduate SE I class at Florida International University (FIU) to obtain feedback on WReSTT V1. Students were administered a survey at the end of the course, and the results indicated that the following changes were needed:

(1) The interface to WReSTT should have more of a social networking tool feel (e.g., Facebook [Facebook Team 2012]);

Table I. Tools in WReSTT V1

Name	Classification	Language	Description
Cobertura	Coverage	Java	Free Java tool that calculates the percentage of code accessed by tests [Cobertura Team 2012]
CppUnit	Test Execution, Unit	C++	C++ unit testing framework [Feathers 2012]
EclEmma	Coverage, Plug-in, Java	Java	Free Java code coverage tool for Eclipse [Hoffmann 2012]
JDepend	Metrics	Java	Tool that traverses Java class file directories and generates design quality metrics for each Java package [Clark 2012]
JUnit	Plug-in, Test Execution, Unit	Java	Unit testing framework for the Java programming language [Gamma and Beck 2012]
SWAT	Test Execution, Web, System/UI	—	Simple Web Automation Toolkit (SWAT) is a library written in C# designed to provide an interface to interact with several different Web browsers [Ultimate Software 2012]
Rational Functional Tester	Test Execution, Web, System/UI	Java, VB .NET	Automated functional and regression testing tool [IBM 2012]

(2) There should be a competition for bonus points between project teams when they access the software testing tutorials and complete the quizzes embedded in the tutorials; and
(3) There should be additional tutorials on the basic concepts of testing.

One of the students in the class, Jairo Pava, was very interested in the WReSTT project and became a member of the development team that coordinated the next version of WReSTT.

Instructor feedback was obtained during two workshops held to introduce CS/IT instructors to software testing concepts and tools, as well as in using the features of WReSTT V1. The workshops, titled *Workshop on Integrating Software Testing into Programming Courses* (WISTPC), were held at FIU during March 2009 [Clarke et al. 2010] and June 2010.[2] The first workshop was attended by 17 instructors from 12 academic institutions and the second workshop by 15 instructors from 12 academic institutions. During the workshops, instructors identified the need to access reports containing (1) student activities in WReSTT, such as which tutorials they were viewing and the comments/questions posted to the forums, and (2) the scores obtained on the quizzes associated with the tutorials.

3.2. WReSTT V2 Design

Based on feedback from students and instructors on WReSTT V1, the second version, WReSTT V2, was developed, which included collaborative learning and classroom management features. The four-tier architecture continues to be used, but the functionality was restructured as shown in Figure 1. The main components in WReSTT V2 are as follows:

—*Authentication*: Provides user with the ability to access the system using their credentials
—*Social*: Allows users to access the social networking features such as creating a user profile, monitoring the activity stream, posting comments to the discussion boards, and monitoring virtual points assigned to users in their respective classes

[2]http://wrestt.cis.fiu.edu/events.

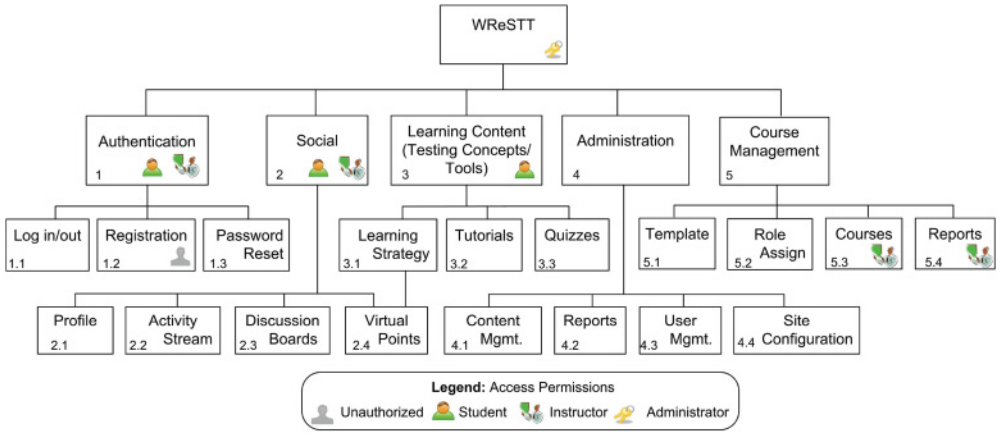


Fig. 1. Block diagram of WReSTT V2 [Clarke et al. 2010, 2012].

—*Learning content*: Contains the tutorials and quizzes on various testing topics and tools to support testing. Note that this component can be replaced for other specific knowledge areas, such as software modeling or programming concepts
—*Administration*: Provides developers and moderators with the ability to update content (e.g., tutorials), generate system-wide reports, monitor and update users' access of the system, and configure the system (e.g., creating new queries to generate new report based on request from instructors)
—*Course management*: Provides administrators with the ability to create new courses and instructors to upload class rolls and generate reports

There is a legend at the bottom of Figure 1 that shows the various categories of users, where the order of increasing access to users is unauthorized, student, instructor, and administrator. Any module inherits the access from its parent and includes new access from that point forward in the hierarchy. Additional aspects of the new structure of WReSTT V2 will be described in the subsequent sections.

Collaborative learning. Based on the feedback from WReSTT V1 users, WReSTT V2 incorporated the concepts of collaborative learning [Smith and MacGregor 1992], online learning communities (OCLs) [Hiltz 1998], and aspects of gamification [Domínguez et al. 2013]. The collaborative learning component in WReSTT V2 is based mainly on student involvement, cooperation, and teamwork throughout the learning process [Smith and MacGregor 1992]. WReSTT V2 primarily achieves student involvement, cooperation, and teamwork by rewarding students with virtual points, requiring team members to collaboratively participate in various activities in a timely manner, and providing opportunities for social engagement. Social engagement includes course discussion forums and activity streams, among other features. Virtual points are awarded for individual activities, as well as team activities where all team members must complete the activity for the team to be awarded points. Although virtual points are independent of the course grade, it is recommended that instructors use the virtual points as a small part of the course grade.

WReSTT V2 provides individual and collaborative learning strategies (see the box labeled 3.1 in Figure 1) that are incorporated in the way users access the testing tutorials and quizzes. Independent of the learning strategy, all students are required to complete all tutorials on testing concepts and tools, and are awarded virtual points based, in part, on the scores obtained on the quizzes. If the user is engaged in a collaborative

strategy, then each student is still required to complete all quizzes; however, quizzes for each team member are generated by selecting appropriate questions from a test bank. Virtual points for quizzes are awarded based on the number of correct answers and the completion time. This is particularly important when using a collaborative strategy, since all the team members are awarded the same number of points, and the points are awarded after the last team member, with respect to time, completes his or her quiz.

In addition to the virtual points awarded to users for the tutorials and quizzes, points are also awarded on an individual basis when users participate in the various social engagement activities. These activities include creating a user profile, which involves uploading a picture; posting questions and comments to the discussion forums; and answering questions posted to the forums. Figure 2 shows one of the student pages from the fall 2011 undergraduate software testing class that used the collaborative learning strategy.

The main student interface is shown in Figure 2 and consists of four sections. The top section shows the tabs to the main components of the Web site. The upper left section of the page shows a picture of the logged-in user (in Figure 2, it is a student from the fall 2012 class, Enmauel Corvo, owner of the page), and below the picture is the sidebar menu. The upper right section of the page shows the pictures of other members of the team, and below the pictures are links to other testing Web sites. The center of the page in Figure 2 is divided into four sections: *top*, provides students with the ability to browse the featured tutorial or browse other tutorials; *middle left*, a list showing the current point leaders in the class; *middle right*, active discussion forum with the most recent entries shown; and *bottom*, the activity stream showing a log of all activities being performed by student in the class.

Collaboration in the OLC supported by WReSTT V2 occurs both at the class level and, as described previously, at the team level. The entire class has real-time access to the active discussions occurring in the forum, where any student in the class can participate in the discussion. In addition, all students in the class can view the activities occurring in WReSTT V2 in real time. These activities include the *Point Leaders*, *Active Discussion,* and *Activity Stream* shown in Figure 2. Note, however, that the focus of the collaboration in WReSTT V2 targets the virtual teams, which may map to actual class project teams.

Course management. The right-most modules of Figure 1—that is, those modules with a prefix of 5—are part of the course management component. These modules are as follows:

—*Template*: Provides the administrator with the ability to create templates for a particular course that can then be used by instructors at various academic institutions;
—*Courses*: Using a given template, instructors can upload the class rolls and assign students to virtual teams; and
—*Report*: Provides instructors with the ability to generate reports for the students in their classes, such as the virtual points for each student in a class, number of times a particular student visited the tutorials, or the time it took the student to complete a specified quiz.

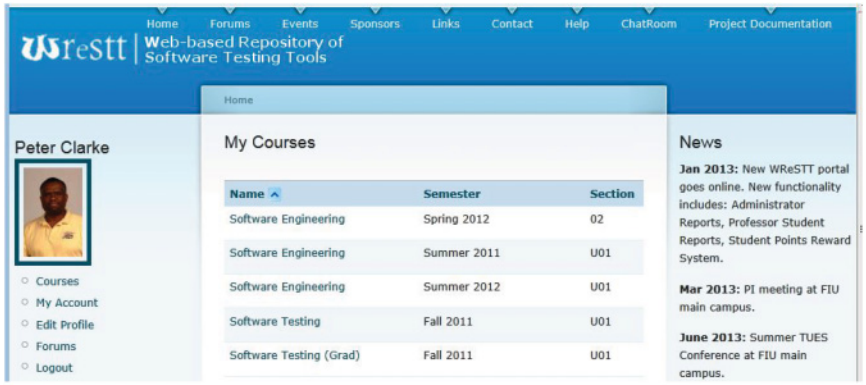
Figure 3 shows two of the pages in the course management component for instructors. Figure 3(a) shows the list of courses that has been assigned to the instructor by the administrator. The top and left sections of the page are similar to the page shown in Figure 2. The list of courses are shown in the center of the page, and the right section shows current news related to WReSTT. Figure 3(b) shows the course management for one of the instructor's courses—in this case, *Software Engineering - Spring 2012, Section : U02*. The functions include importing/exporting class rolls, viewing student

Fig. 2. A student's home page in WReSTT V2.

reports, creating virtual teams, allocation of virtual points to each task, and viewing the usage statistics for each student.

3.3. Using WReSTT in the Classroom

WReSTT can be used in a variety of ways based on the level of testing that is required in the course. The authors have experience using WReSTT in SE and software testing courses both at the graduate and undergraduate levels. In this section, we describe how WReSTT could be used in SE courses similar to how it was used in the study presented in the next section.



(a)

(b)

Fig. 3. Course management pages. (a) List of instructor's courses. (b) Instructor's course management page.

WReSTT may be introduced in an undergraduate SE course using a minimally disruptive approach where instructors do not have to significantly change the content of their syllabi or their teaching styles. Following is a summary of the steps an instructor may take during the semester when using WReSTT in a class:

(1) Prior to the beginning of the semester, the instructor modifies his or her course grading scheme to include some allocation of points for the virtual points awarded by WReSTT to each student in the class.
(2) Early in the semester, the instructor introduces students to the features of WReSTT and assigns them to virtual teams.
(3) For some set of assignments, the instructor requires students to submit a report with each assignment containing the testing techniques used, test cases, test coverage achieved using different criteria, and an explanation of any criteria for which the specified coverage is not achieved.
(4) During the semester, the instructor monitors the postings to the forums and provides feedback to the virtual teams and the class, with respect to their participation on completing the tutorials and quizzes.

Virtual points. Points are awarded to students in the class based on their individual effort and/or their team effort after the completion of various tasks. The individual points may be awarded for the following: (a) updating the student profile (e.g.,

uploading a picture), (b) posting at least one question/comment related to testing concepts and testing tools to the forum, and (c) completing a tutorial and quiz set with at least some predefined percentage (xx%) of correct answers. Note that there may be multiple tutorial quiz sets assigned to the class. The team points are awarded based on the time each team completed a tutorial and quiz set (at least xx% correct answers) when compared to other teams. The instructor is free to assign any value for the preceding points by clicking on the *Manage* option under *Points Rewards* in the course management screen shown in Figure 3(b).

WReSTT introduction. Depending on when the instructor plans to use testing in his or her class (e.g., the introduction of the testing topics or the due dates for programming assignments), the instructor can upload the class rolls 4 weeks prior to that time, assign virtual teams, and briefly introduce students to the features in WReSTT. It is also very important to explain how the virtual points will be awarded and how they will be integrated into the course grade.

Assignments. In our SE courses, students are required to submit code at the end of the semester as part of the final deliverable. Students are also required to demonstrate the running software, and explain and demonstrate how the testing tools were used, with respect to test automation and code coverage. The final project report includes a description of the test plan (test cases, procedures used, and test results), testing tools used, and code coverage achieved. If the required coverage is not achieved, students provide an explanation for why this is the case. Due to the time allocated to the testing in a SE course, it is not feasible to ask students to update the test set to improve coverage and resubmit the project. Updating the test set to improve code coverage works very well in software testing courses, since more time is spent on the different approaches to testing.

Monitor teams. The instructor can monitor students' interaction with WReSTT by generating reports either on a per-student basis or for the entire class. The reports highlight those students who have not completed a specific tutorial or quiz. These reports are generated using the *View* option under *Student Reports* in the course management screen shown in Figure 3(b). Our experience has shown that using the collaborative learning approach encourages team members to complete the tutorial and quiz sets since the entire team benefits from the bonus points.

An updated WReSTT V2 is available to the public, and WReSTT V1 is now retired. Instructors may request access as an *instructor* so that they may use WReSTT in their classes by contacting the WReSTT team (http://wrestt.cis.fiu.edu/).

4. EMPIRICAL STUDY

Over the past 3 years, we have conducted several studies to determine the impact WReSTT has on student learning. Included were initial studies to assist us in determining the evolution of WReSTT [Clarke et al. 2010, 2011] and a preliminary study evaluating both versions of WReSTT [Clarke et al. 2012]. In this section, we motivate the study and identify the objectives of the study; describe the subject selection, experimental design, and instrumentation; and present a detailed statistical analysis and discussion of the results, including a validity evaluation [Basili et al. 1986; Trochim 2001; Wohlin et al. 2000].

4.1. Research Objectives

In Section 2.1, we identified several of the challenges of teaching software testing in an SE course and other upper-level courses with a major programming component. The overriding challenge is that there are usually too many topics to be covered in

Table II. Sample Used in the Study

Group ID	Class	Pre/Posttest			WReSTT Survey		
		Enrollment	Participation	%	Enrollment	Participation	%
	Control Group:						
CG1	CEN4010 SP11	36	18	50.0	—	—	—
CG2	CEN4010 SP12	32	17	58.6	—	—	—
	Treatment Group:						
TG1	CEN4010 SU11	25	22	88.0	25	23	92
TG2	CEN4010 SU12	15	15	100	15	15	100
CEN4010—Software Engineering I: SP1x—Spring 201x; SU1x—Summer 201x							

these courses, and there is little or no time to teach testing. The WReSTT project was developed to address this challenge by providing a minimally disruptive approach to integrating software testing into SE and upper-level programming courses. The key aspect of the project is that students supplement their classroom instruction by accessing learning materials on testing concepts and testing tools available in WReSTT. WReSTT has evolved from a simple repository providing tutorials on testing tools only to a learning environment that contains tutorials and quizzes on testing concepts and supports collaborative learning integrated with some social networking activities.

Based on our initial studies [Clarke et al. 2010, 2011, 2012], there are two factors that affect the success of any approach that uses WReSTT to support learning. These factors are as follows: (1) the approach should be minimally disruptive (i.e., requires instructors to make little changes to the content of their syllabi or their teaching styles), and (2) the approach should be attractive to students and motivate them to engage the materials provided by WReSTT. The study presented in this section focuses on the second factor—that is, students' use of WReSTT. The objectives of the study are as follows:

—*Objective 1*: Determine if students who use WReSTT as part of their undergraduate SE course will have greater *general knowledge of software testing* than students who do not use WReSTT.
—*Objective 2*: Determine if students who use WReSTT as part of their undergraduate SE course will have greater *general knowledge and skills in using software test generation techniques* than students who do not use WReSTT.
—*Objective 3*: Determine if students who use WReSTT as part of their undergraduate SE course will have greater *general knowledge of software testing tools* than students who do not use WReSTT.
—*Objective 4*: Determine if students who use WReSTT will find it *engaging and enjoyable to use*.
—*Objective 5*: Determine if students who use WReSTT will be more likely to employ the use of *software testing tools in their SE team project* than students who do not use WReSTT.

As stated previously, the time dedicated to testing in the SE course is limited; therefore, we do not expect students to iteratively improve their test suites based on the code coverage results. However, we expect them to have an understanding of the effectiveness of the test suite based on the results of code coverage.

4.2. Methods

Sample. Students from four SE I classes participated in the study. The classes were (1) *CEN4010 SP11*—Spring 2011, (2) *CEN4010 SU11*—Summer 2011, (3) *CEN4010 SP12*—Spring 2012, and (4) *CEN4010 SU12*—Summer 2012. Table II shows a summary of the classes that participated in the study. A total of 72 subjects participated,

including two control groups, CG1 and CG2, containing 18 and 17 students, respectively, and two treatment groups, TG1 and TG2, containing 22 and 15 students, respectively. There was no difference in terms of course preparation and demographics between the subjects in the control groups and treatment groups.

Measurement. Data for the study was collected using (1) two instruments, (2) artifacts from the team projects, and (3) logs from WReSTT. The two instruments were a pre/posttest (see Appendix A) and a WReSTT survey (see Appendix B). The pre/posttest was designed to identify students' knowledge of testing prior to being taught the testing topic in the SE class and being exposed to the learning materials in WReSTT (treatment). It was again used to determine students' knowledge after being exposed to the class instruction and the treatment. The pre/posttest shown in Appendix A also contains sample answers or guidelines for the answers to several of the questions, shown in bold italics.

The pre/posttest, shown in Appendix A, consists of eight questions described as follows:

—Q1 focused on the objective of program testing.
—Q2 involved the identification of testing techniques and writing test cases for a Java method.
—Q3 and Q4 focused on the use of testing tools, identification of tools, and the classifications of tools for unit testing, functional testing, and code coverage.
—Q5 and Q6 focused on the knowledge of online testing resources and the type of materials contained in the resources.
—Q7 and Q8 addressed the importance of using testing tools while working on programming assignments.

The pre/posttest used in the preliminary studies prior to spring 2011 (the first control group) [Clarke et al. 2010, 2011] was considered to be too subjective, so Q2 was added. Note that the answers/guidelines for Q2(b) are broken down into four parts to simplify the evaluation of the pre/posttest.

The WReSTT survey instrument, shown in Appendix B, consists of 30 questions divided into five groups. These groups are described as follows:

—Q1 focused on the use of testing resources other than WReSTT.
—Q2 through Q15 focused on the overall reaction to the two versions of the WReSTT Web sites. These questions were adapted from page 140 of Tullis and Albert [2008].
—Q16 through Q21 obtained feedback on the impact that using WReSTT had on learning testing concepts and testing tools.
—Q22 through Q26 obtained feedback on the impact that the collaborative learning component in WReSTT V2 had on visiting the Web site and participation in team activities.
—Q27 through Q30 provided the opportunity for feedback on WReSTT V1 and V2 using the open-ended question format.

Q2 through Q26 were answered using the following Likert scale: *1 = Strongly Disagree*, *2 = Disagree*, *3 = Neither Agree nor Disagree*, *4 = Agree*, *5 = Strongly Agree*.

The artifacts for the team project include a written system's manual, a user's guide, and the source code. In the testing section of the system's manual, students are required to document the testing process, which includes the subsystems and systems test cases, results of executing the test, and a description of any testing tools used. Each team was required to demonstrate the execution of their software project at the end of the semester and, if any testing tools were used, to demonstrate how the tools worked. This demonstration was used to confirm the answers they provided during the posttest and to evaluate their practical testing skills for the course.

Design. The pre/posttest instrument was administered to the four different groups shown in Table II. The main topics in the CEN4010 course are Requirements Elicitation, Requirements Analysis, Systems Design, Detailed Design, and Testing. The students are evaluated based on two examinations and a semester team project consisting of three deliverables. During the final deliverable, students are required to present a live demonstration of the system, which includes demonstrating the use of any tools used to support testing during development.

The pretest was administered prior to the testing part of the CEN4010 course (i.e., prior to week 8 of the semester), and the posttest was administered at the end of the semester. Before administering the pretest, students were informed that the study would not impact their grades and were not required to take the test. The control (spring) and treatment (summer) groups were given the same class notes and used the same textbook throughout the course. The only differences were (1) the instructors, (2) the summer semester was 2 weeks shorter than the spring semester, and (3) the treatment groups were exposed to WReSTT prior to the testing topic being taught in the course. The answers/guidelines for the questions, shown in bold italics in Appendix A, were used in the evaluation of the tests. The points allocated to each correct answer are shown in italics above each answer.

The survey instrument was administered *only* to the classes in the treatment groups (CEN4010 SU11 and CEN4010 SU11). Students in the treatment groups were introduced to the two versions of WReSTT during weeks 8 (WReSTT V1) and 11 (WReSTT V2) of the semester, respectively. Students were required to register on WReSTT V1 individually, whereas the instructor registered students in WReSTT V2 based on the teams to which they were assigned for the team projects. The project teams were the same as the virtual teams in the study (note, however, that this did not have to be the case). Students in the control groups were not exposed to WReSTT V1 or V2.

Prior to providing students access to WReSTT V2, they were informed of how the virtual points would be awarded, which were on an individual as well as a team basis. The individual points were awarded as follows: (a) updating the student profile (e.g., uploading a picture), *1 point*; (b) posting at least one question/comment related to testing concepts and testing tools to the forum, *1 point*; and (c) completing the tutorial and quiz with at least 80% correct answers, *1 point*. The team points were awarded based on the time in which each team completed the tutorial and quiz (with at least 80% correct answers) when compared to other teams. The points awarded to members of the team completing the tutorial and quiz first was *8 points*, second was *5 points*, and third was *5 points*. The data used to allocate the virtual points to each student was obtained by querying the WReSTT system logs. We also cross-checked the content of the data logs with the responses in the WReSTT survey to see if there was a correlation.

At the end of the semester, the team project artifacts were reviewed and archived. The artifacts were reviewed by at least two SE course instructors participating in the study. The review process included checking the system's manual and the source code to identify the following: implementation language; if automated testing was used; the unit, functional, and code coverage tool(s) used for automated testing; and the recorded percentage of code coverage achieved. To support the claims made in the system's manual, students were required to demonstrate the tools used during the final project presentation.

4.3. Results and Analysis

Pre/posttest. To evaluate the effects of the use of WReSTT on students' knowledge and understanding of software testing techniques and tools, students were administered a pre- and posttest software testing assessment prior to the testing topic being taught and at the end of each semester, respectively. For each assessment, students were

Table III. Overall Results of Pretest and Posttest

			Pretest		*Posttest*	
	Group	*N*	*Mean*	*SD*	*Mean*	*SD*
(A)	Control	35	12.83	8.41	11.29	8.91
	Treatment	37	10.97	4.99	21.70	6.82
	Measure	*Main Effects*				
	Test (2)	$F(1,70) = 27.82$, $p < 0.01$, students performed better on the posttest Follow-up analysis: $t(37) = -8.15$, $p < 0.01$, shows significant difference between pre- and posttest scores in the treatment group, but not in the control group				
(B)	Group (2)	$F(1,70) = 8.00$, $p < 0.01$, students performed better in the treatment group				
	Measures	*Interaction*				
(C)	Group, Test	$F(1,70) = 49.64$, $p < 0.01$, significant interaction				

Note: Section (A) reveals students' mean scores and standard deviations on the software testing pre- and posttests in the control and treatment groups. Section (B) presents the main effects of the the test and group using a 2 (test group) × 2 (test) repeated measures ANOVA. Section (C) presents the interaction between the test group and the test.

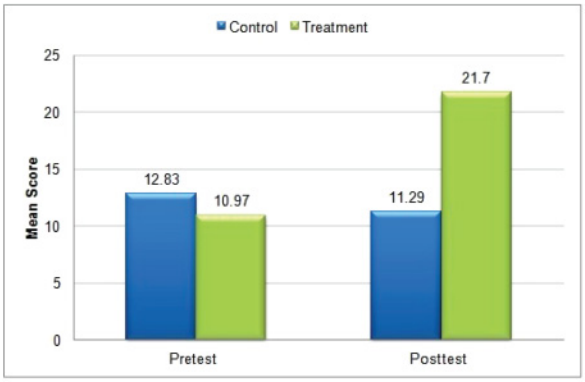


Fig. 4. General software testing knowledge mean scores.

given a score indicating their performance on the test. Preliminary results found no significant differences in test scores between the two control groups (CG1 and CG2) or between the two treatment groups (TG1 and TG2). Thus, participants were grouped according to test group (control vs. treatment groups).

Tables III through VI show a summary of the results and analysis for the pre/posttest for the control and treatment groups. Each table contains three sections: Section (A) shows the mean and standard deviation for the score on the test; Section (B) shows the main effects of the test using a 2 (test group) × 2 (test) repeated measures ANOVA [Neter et al. 1990]; and Section (C) shows the interaction between the test group and the test.

Table III and Figure 4 show the overall results of the pretest and posttest administered to students in the study. Students in the treatment group, who used WReSTT, performed significantly better on their posttests than students in the control group, who did not use WReSTT. There was no significant difference between their pretest scores, indicating that this difference was not due to differences in students' knowledge before taking the course or the semester. To look more closely at student achievement, the pre- and posttests were broken down into three general knowledge areas: testing technique knowledge, tool usage, and tool proficiency level. Analyses were then performed on each of these areas.

Table IV. Summary of Results—Testing Technique Knowledge

			Pretest		*Posttest*	
	Group	*N*	*Mean*	*SD*	*Mean*	*SD*
(A)	Control	35	4.77	3.71	3.82	3.45
	Treatment	37	5.22	3.68	6.43	3.78
	Measure	*Main Effects*				
(B)	Test (2)	$F(1,70) = 4.17$, $p < 0.05$, students from the treatment group, but not the control group, tended to perform better on the posttest than the pretest Follow-up analysis: $t(37) = -1.92$, $p = 0.06$, shows no significant difference, between pre- and posttest scores in the treatment group, similarly for the control group				
	Group (2)	$F(1,70) = 4.17$, $p < 0.05$, students performed better in the treatment group				
	Measures	*Interaction*				
(C)	Group, Test	$F(1,70) = 4.172$, $p < 0.05$, significant interaction				

Note: Section (A) reveals students' mean scores and standard deviations on the testing technique knowledge pre- and posttests in the control and treatment groups. Section (B) presents the main effects of the the test and group using a 2 (test group) $\times$ 2 (test) repeated measures ANOVA. Section (C) presents the interaction between the test group and the test.

Testing technique knowledge. To indicate the level of each student's knowledge of testing techniques, a score was calculated by adding scores on Q1, Q2(a), and Q2(b) (see Appendix A). The results in Table IV indicate that the use of WReSTT as a teaching tool for students may improve their knowledge and use of software testing techniques. When comparing the pre- and posttest scores of the treatment group alone, the differences were close but did not quite reach significance ($p = 0.06$). Table IV Section (B) shows a follow-up analysis using the two-test measure.

Testing tool usage. As can be seen in Table V and Figure 5, results indicate that the use of WReSTT as a teaching tool for students can improve their knowledge and use of software testing tools. These results were computed based on the answers for Q3, Q4(a), and Q4(b) (see Appendix A). There was no significant difference between the treatment and control group pretest scores, indicating that this difference was not due to differences in students' knowledge of testing tools before taking the course.

Tool proficiency confidence. One of the subquestions that we were interested in exploring was whether the use of WReSTT would improve not only students' knowledge and use of software testing tools but also their perceived proficiency in using these tools (see Q4(b) in Appendix A). As can be seen in Table VI and Figure 6, students in the treatment group reported greater software testing tool proficiency levels on their posttests than students in the control group, as well as a significant increase in proficiency from their pretests.

WReSTT user experience survey. Students in the treatment groups were additionally administered a survey at the end of the semester designed to assess their perception of the ease of use and usefulness of WResTT. A total of 39 students participated in the survey. In Section 1 (Q1) of the survey (see Appendix B), 31% of students ($N = 12$) indicated that they had previously used a learning resource other than WReSTT to learn about software testing, whereas 44% ($N = 17$) indicated no use of additional resources (26% of students, $N = 10$, did not respond to the question). Students in the control groups were not administered the WReSTT survey, as using WReSTT was part of the treatment and they would not have been exposed to any version of WReSTT.

Table V. Summary of Results—Testing Tool Usage

			Pretest		*Posttest*	
	Group	*N*	*Mean*	*SD*	*Mean*	*SD*
(A)	Control	35	0.66	1.85	0.74	2.09
	Treatment	37	0.38	1.14	2.81	1.58
	Measure	*Main Effects*				
(B)	Test (2)	$F(1,70) = 97.57$, $p < 0.01$, students performed better on the posttest Follow-up analysis: $t(36) = -10.25$, $p < 0.01$, shows significant difference between pre- and post-test scores in the treatment group, but not the control group				
	Group (2)	$F(1,70) = 5.60$, $p < 0.05$, students performed better in the treatment group				
	Measures	*Interaction*				
(C)	Group, Test	$F(1,70) = 84.74$, $p < 0.01$, significant interaction				

Note: Section (A) reveals students' mean scores and standard deviations on the tool usage pre- and posttests in the control and treatment groups. Section (B) presents the main effects of the the test and group using a 2 (test group) × 2 (test) repeated measures ANOVA. Section (C) presents the interaction between the test group and the test.

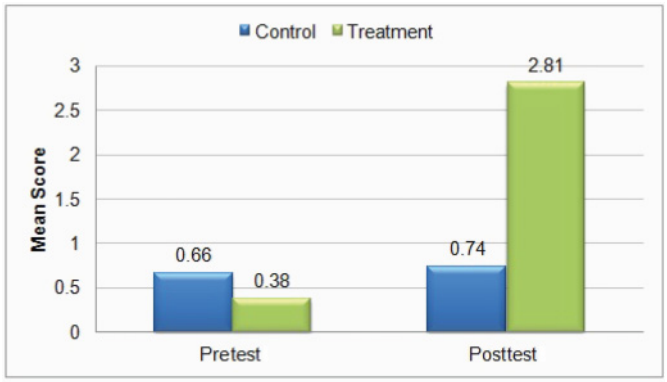


Fig. 5. Testing tool usage mean scores.

Overall reaction of WReSTT V1 and V2. Students' overall reactions to both versions of WReSTT were positive, with mean scores on questions in Section 2 (Q2 through Q15) of the survey above 3.18 for V1, and above 3.62 for V2. Mean scores (and standard deviation) for each question in Section 2 for both versions of WReSTT are shown in Columns 4 and 5 of Table VII. Figure 7 shows the bar chart comparison of the mean scores for the two versions of WReSTT. To compare student reactions to V1 and V2 of WReSTT, students' responses to each question in Section 2 were analyzed using the Wilcoxon signed-ranks test [Siegel and Castellan 1988]. The results, shown in Column 6 of Table VII, revealed that except for Q8 and Q9, there was a significant difference between students' reactions to the two versions of WReSTT, where students indicated a preference for WReSTT V2 over V1.

Perceived usefulness of WReSTT. As can be seen in Table VIII, students' perceptions of the usefulness of the tutorials available in WReSTT were also positive, with mean scores above 3.38 for Q16, Q17, and Q18 in Section 3 (Q16 through Q21). Students' responses to Q21, asking whether they would have used testing tools if WReSTT had not existed, indicated that many would not ($M = 2.92$, $SD = 1.3$).

Table VI. Summary of Results—Perceived Testing Tool Proficiency

			Pretest		*Posttest*	
	Group	*N*	*Mean*	*SD*	*Mean*	*SD*
(A)	Control	35	2.67	4.51	2.77	4.43
	Treatment	37	0.57	1.35	4.30	3.12
	Measure	*Main Effects*				
(B)	Test (2)	$F(1,70) = 21.26$, $p < 0.01$, students performed better on the posttest Follow-up analysis: $t(36) = -10.25$, $p < 0.01$, shows significant difference between pre- and posttest scores in the treatment group, but not the control group				
	Group (2)	$F(1,70) = 7.46$, $p < 0.01$, students performed better in the treatment group				
	Measures	*Interaction*				
(C)	Group, Test	$F(1,70) = 19.39$, $p < 0.01$, significant interaction				

Note: Section (A) reveals students' mean scores and standard deviations on the perceived tool proficiency pre- and posttests in the control and treatment groups. Section (B) presents the main effects of the the test and group using a 2 (test group) × 2 (test) repeated measures ANOVA. Section (C) presents the interaction between the test group and the test.

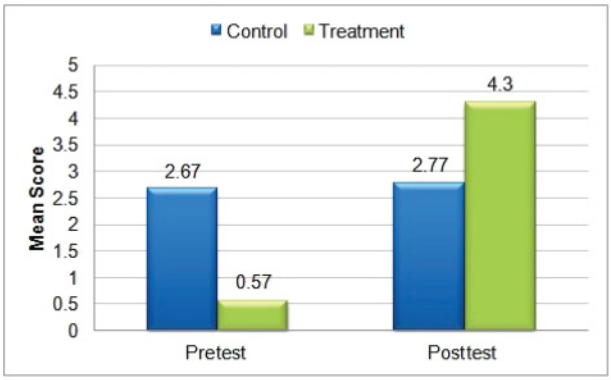


Fig. 6. Perceived tool proficiency mean scores.

Perceived usefulness of collaborative learning environment. As can be seen in Table IX, students indicated that the use of a collaborative learning environment in WReSTT was a motivating factor in their learning, with mean scores above 3.76 on Q22 through Q25 in Section 4 (Q22 through Q26) of the survey. They found that the collaborative learning environment in WReSTT motivates them to visit the site and complete assignments. As can be seen in Figure 8, students indicated at high levels that the use of virtual points and event streaming encouraged both them and their fellow team members to complete tasks in WReSTT.

In an attempt to validate the survey data obtained from the treatment groups, we decided to review how the virtual points were allocated in the two SE classes. Note that the virtual points were only allocated for the interaction with WReSTT V2. In the SE summer 2011 class (TG1), all students (23) who took the survey received virtual points. Based on the logs of the system, the virtual points were distributed as follows: all students read the tutorial, and all students achieved a grade higher than 80% on the quiz; 13 students posted messages to the forum with an average of 4 messages per student; and all students uploaded a picture to WReSTT. For the SE summer 2012 class (TG2), all students (15) who took the survey received virtual points. The distribution of the points were as follows: 14 students read the tutorial, and 13 students achieved a

Table VII. Students' Mean Scores (Standard Deviations) in Section 2 of the Survey Measuring Their Overall Reactions to WReSTT V1 and V2

Q	*Overall Reaction to Web Sites*	*N*	*Version 1 M (SD)*	*Version 2 M (SD)*	*Wilcoxon Signed-Ranks Test*
2	Overall, I am satisfied with how easy it is to use the Web site.	38	3.63 (1.0)	4.15 (0.9)	$z = -2.84$, $p < 0.01$
3	It is simple to use the Web site.	38	3.76 (1.0)	4.15 (0.9)	$z = -2.78$, $p < 0.01$
4	I feel comfortable using the Web site.	39	3.71 (1.0)	4.18 (0.7)	$z = -2.80$, $p < 0.01$
5	It was easy to learn to use the Web site.	39	3.84 (1.0)	4.29 (0.8)	$z = -3.05$, $p < 0.01$
6	I believe I became productive quickly using the Web site.	39	3.65 (0.9)	3.97 (0.9)	$z = -2.18$, $p < 0.05$
7	The information (such as online help, on-page messages, and other documentation) provided with the Web site is clear.	37	3.53 (1.0)	3.86 (1.0)	$z = -2.13$, $p < 0.05$
8	It is easy to find the information I need.	39	3.46 (1.0)	3.62 (0.9)	—
9	The information is effective in helping me complete the tasks and scenarios.	38	3.78 (1.0)	4.03 (0.9)	—
10	The interface of the Web site is pleasant.	38	3.53 (1.1)	4.29 (1.0)	$z = -3.89$, $p < 0.01$
11	I like using the interface of this Web site.	39	3.32 (1.0)	3.92 (1.0)	$z = -3.51$, $p < 0.01$
12	The Web site has all of the functions and capabilities that I expect it to have.	39	3.37 (1.0)	3.69 (1.1)	$z = -2.07$, $p < 0.05$
13	I believe that the Web site helped me earn a better grade.	38	3.56 (1.0)	3.95 (0.9)	$z = -2.49$, $p < 0.05$
14	I would recommend the Web site to fellow students.	39	3.71 (0.8)	4.15 (0.8)	$z = -2.79$, $p < 0.01$
15	Overall, I am satisfied with the Web site.	39	3.63 (0.8)	4.13 (0.9)	$z = -3.00$, $p < 0.01$

grade higher than 80% on the quiz; 11 students posted messages to the forum with an average of 1.5 messages per student; and all students uploaded a picture to WReSTT.

Review of team artifacts. Table X shows a summary of the data collected after reviewing the team project artifacts for the control and treatment groups in the study. The seven columns of the table represent the team number, the implementation language for the project, the unit testing tools used, code coverage tools used, number of test cases written (system and subsystem), and percentage code coverage achieved. The rows in the table are divided into four groups, the top two for the control groups and the next two for the treatment groups.

As shown in Table X, none of the teams in CG1 used tools during the testing phase of their projects, and only one team in CG2 used JUnit during testing. In contrast, both treatment groups used tools to test the code they developed, with all five teams in TG1 and two out of four in TG2 using testing tools in their projects. Three teams from the treatment groups used code coverage tools to check the quality of their test suites. It is interesting to note that students used testing tools for which there were no tutorials in WReSTT. These tools included *Visual Studio Unit Testing Tool* [Microsoft Corporation 2013], *PHPUnit* [Bergmann 2013], *OCUnit* [Sen:te 2013], and *Xcode CoveryStory* [CoverStory Team 2013].

Team 5 of TG1 wrote 18 system test cases with a pass rate of 89% (16 of 18 passing) and 9 subsystem test cases with 100% pass rate. The code coverage reported by Cobertura for the subsystem test cases was 93% or 103/110 LOC and 79% or 27/34 branches for the storage component of the system. These results formed 16% of 682 LOC and 17% of 152 branches for the entire system. Note that students were instructed to test only one subsystem due to time allocated to testing in the software project. The results for TG2 Teams 2 and 3 can be explained in a similar manner to that of TG1 Team 5.

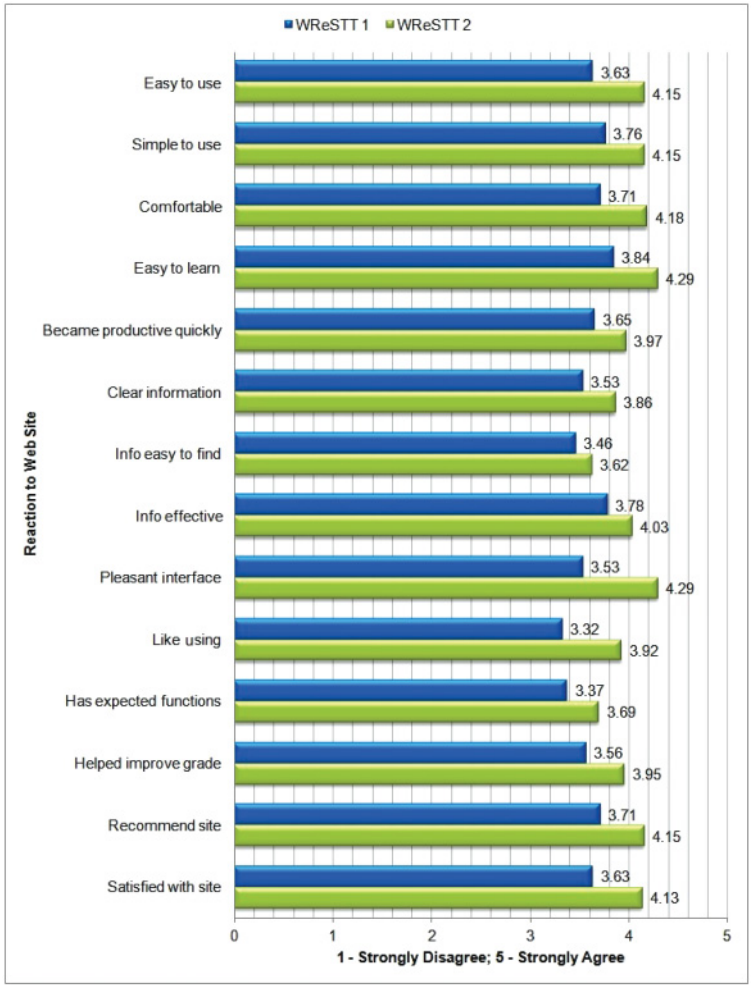


Fig. 7. Overall reaction to WReSTT V1 and V2 mean scores.

That is, using 11 test cases, Team 2 obtained 68.8% of 648 LOC during subsystem testing, and using 4 test cases, Team 4 obtained 32.4% of 689 LOC.

4.4. Discussion

Objective 1. Determine if students who use WReSTT as part of their undergraduate SE course will have greater *general knowledge of software testing* than students who do not use WReSTT.

The results shown in Table III and Figure 4 confirm and extend previous findings showing improved student understanding and knowledge of software testing techniques and tools by using WReSTT. The current studies involve students from four different semesters (two control and two treatment) over the period of a year. The consistency of the results across the groups regardless of the semester provide further support that the findings are the result of the use of WReSTT itself and not an anomaly of a particular semester or group of students. Thus, the use of WReSTT can increase students' general knowledge of software testing.

Table VIII. 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.	39	4.38 (0.7)
17	The tutorials in WReSTT helped me to better understand how to use unit testing tools.	39	4.00 (1.1)
18	The tutorials in WReSTT helped me to better understand how to use code coverage testing tools.	39	3.38 (1.4)
19	The tutorials in WReSTT helped me to better understand how to use functional testing tools.	39	2.87 (1.7)
20	The number of tutorials in WReSTT is adequate.	39	3.56 (1.0)
21	I would have used testing tools in my project if WReSTT did not exist.	39	2.92 (1.13)

Table IX. Students' Mean Scores (Standard Deviations) of the Survey Measuring Usefulness of the Collaborative Learning Environment in WReSTT

Q	*Collaborative Learning–Related Questions*	*N*	*M (SD)*
22	The use of virtual points in WReSTT V2 encouraged me to visit the Web site and complete the tasks.	39	4.54 (0.8)
23	The use of virtual points in WReSTT V2 encouraged my team to visit the Web site and complete the tasks.	39	4.56 (0.7)
24	The event stream showing the activities of the other members in the class encouraged me to complete my tasks in WReSTT V2.	39	4.26 (0.9)
25	The event stream showing the activities of the other members in the class encouraged my team to complete my tasks in WReSTT V2.	39	4.38 (0.8)
26	Our team devised a plan to get the maximum number of points in WReSTT V2.	38	3.76 (1.4)

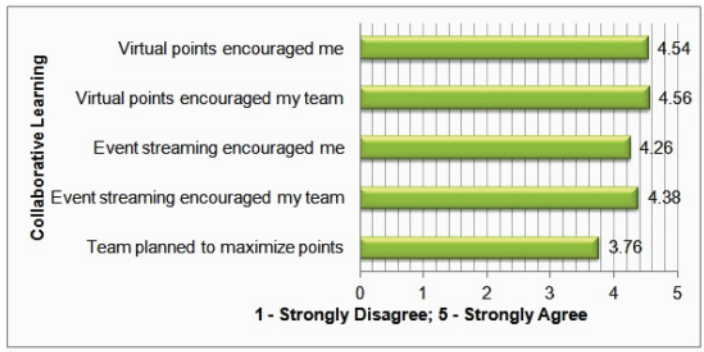


Fig. 8. Perceived usefulness of collaborative learning environment mean scores.

In addition to looking at students' general understanding of software testing, we were interested in taking a more detailed look at students' knowledge and use of software test generation techniques (Objective 2) and tools (Objective 3). Looking at these areas more closely can provide a more fine-tuned assessment of the efficacy of use of the system and better inform the future direction of improvements for increased student learning. It also helps to provide a better understanding of the depth of knowledge gained by students who use the system.

Objective 2. Determine if students who use WReSTT as part of their undergraduate SE course will have greater *general knowledge and skills in using software test generation techniques* than students who do not use WReSTT.

Table X. Review of the Team Artifacts Showing Those Teams That Used Testing Tools

Team	*Language*	*Unit Testing Tools*	*Coverage Tools*	*# Test Cases (P/F)*		*% Code Coverage*
				System	*Subsystem*	
Control Groups:						
CG1: Spring 2011—5 teams (no testing tools used)						
CG2: Spring 2012—4 teams (3 of the teams did not use testing tools)						
1	Java	JUnit		8 (8/0)	16 (16/0)	
Treatment Groups:						
TG1: Summer 2011—5 teams						
1	Java	JUnit		21 (21/0)	8 (8/0)	
2	C#	Visual Studio Unit Testing Tool		18 (16/2)	9 (9/0)	
3	Java, PHP	JUnit, PHPUnit		18 (18/0)	20 (20/0)	
4	Java	JUnit		18 (12/6)	26 (NR)	
5	Java	JUnit	Cobertura	18 (16/2)	9 (9/0)	93% of 110 LOC, 79% of 34 branches
TG2: Summer 2012—4 teams (2 of the teams did not use testing tools)						
2	Objective-C	OCUnit	Xcode/ CoverStory	27 (27/0)	11 (11/0)	68.8% of 648 LOC
3	Objective-C	OCUnit	Xcode/ CoverStory	24 (24/0)	4 (4/0)	32.4% of 689 LOC

NR, Not Reported.

As can be seen in Table IV, the pretest scores of the treatment groups were slightly, although not significantly, higher than those of the control groups. Further, the posttest scores of the control groups were actually slightly lower than their pretest scores, although, again, it was not a significant difference. Looking at this superficially, one might suggest that this indicates there might have been some initial difference between the two groups. As those differences were not significant, we reject that argument.

Nevertheless, these possibly conflicting findings indicate that although the use of WReSTT may improve students' knowledge of testing techniques, further research should be conducted. Further study or more refined measures may shed more light in this area. Alternately, it may be that more content in this area should be added to WReSTT or that changes to how the resources currently in WReSTT are used might help students improve their knowledge in this area.

Objective 3. Determine if students who use WReSTT as part of their undergraduate SE course will have greater *general knowledge of software testing tools* than students who do not use WReSTT.

Students in the treatment groups exhibited greater understanding and increased use of software testing tools on their posttests than students in the control groups, as well as a significant increase in knowledge from their pretests. These results are reported in Table V and Figure 5, and supported by the results in Table X.

Interestingly, students in the treatment groups reported significantly lower confidence of proficiency in test tool usage at the beginning of the semester than students in the control groups. It is not clear why this is the case, but the significant increase in their reported proficiency from before their use of WReSTT to after their use of WReSTT raises some interesting questions for future research. It may be, for example,

that having a very low perception of one's proficiency in testing tool usage itself may be an influencing factor in how much effort is put toward increasing that proficiency. Future studies could help answer this question.

Based on our results we cannot conclusively say that the use of WReSTT improves students' proficiency in using software testing tools, only that it increases their confidence in using the testing tools. Given that most of the students are complete novices when it comes to using testing tools, it is likely that the use of WReSTT does contribute to improving actual proficiency.

Objective 4. Determine if students who use WReSTT will find it *engaging and enjoyable to use*.

Results indicate that students do find WReSTT an engaging and enjoyable learning resource for software testing techniques and tools. Specifically, students indicated that the tutorials in WReSTT helped them understand both software testing concepts and tools, and that there are a sufficient number of tutorials in WReSTT. Students also indicated that both versions of WReSTT helped them complete course tasks and earn a better grade in the class. Regarding ease of use, students indicated that both versions of WReSTT are easy to use and pleasant and would recommend use of the sites to other students.

Overall, students' responses indicate that they have a more positive view of WReSTT V2 than WReSTT V1. As can be seen in Table VII and Figure 7, for all but two of the questions regarding the ease of use and usefulness of the sites, students rated WReSTT V2 significantly higher than WReSTT V1. The only two areas in which there was no difference in student responses was in ease of finding needed information and the effectiveness of the information available helping them complete their tasks and scenarios. Using the results from this study, WReSTT V2 has been updated to reflect the preference of the students.

Students do find that the use of a collaborative learning environment in WReSTT motivates them to visit the site and complete assignments. As can be seen in Table IX and Figure 8, students indicated at high levels that the use of virtual points and event streaming encouraged both them and their fellow team members to complete tasks in WReSTT.

Objective 5. Determine if students who use WReSTT will be more likely to employ the use of *software testing tools in their SE team project* than students who do not use WReSTT.

As can be seen in Table X, students who used WReSTT tended to use testing tools in their team project more often than students who did not use WReSTT. Of the nine teams in the control groups, only one team used a testing tool in their class project, whereas seven out of the nine teams in the treatment groups used testing tools. The table also shows the number of test cases used during system and subsystem testing, including the number that passed and failed. Of the seven teams that used testing tools, three reported code coverage results obtained during subsystem testing as shown in Table X. Two of the teams in the TG2 (summer 2012) did not complete the project and did not report usage of the testing tools.

Students were required to demonstrate their knowledge of testing tools during the final class presentation as a means to validate their use of the testing tools. Based on the observed presentations, students demonstrated that they understood how to use the various features in the JUnit framework, including setting up and tearing down test environments, creating stubs and drivers, and explaining why some parts of the test code were not covered. It should be noted that one team in the control group that used testing tools included a student who took the software testing class in a previous semester.

A review of the artifacts containing testing code provided further evidence that students who used the testing tools were able to write more complete test cases, as reflected in the system documentation. In addition, these students were able to use the assert statements, based on the expected results of the test case, to determine if the test passed or failed. They also used one test class in most cases as the test driver for subsystem testing, making use of the test setup and tear down methods provided by the testing framework.

Students who did not use testing tools used "print" statements and integer codes to determine if test cases passed or failed. For example, during unit testing, Team 3 from CG2 (spring 2012) returned a series of numeric codes to signal the correctness of the test cases, such as –2 incorrect input and –3 connection error, as well as used "print" statements to echo output to the screen. Clearly, this approach to unit testing is more time consuming, especially during regression testing.

Students were not only more knowledgeable about software testing, but because of the socially competitive nature of the Web site, students were also more willing to learn about and apply their testing knowledge even though they were not required to do so in their projects. This willingness to learn more was reflected in the fact that some students use testing tools for which there are currently no tutorials in WReSTT, such as *Xcode CoveryStory* [CoverStory Team 2013].

Threats to validity. The threats to validity of this study are related mainly to the selection of the samples for the control and treatment groups, and the instructors who were assigned to the study. Table II shows the sample size used in the study, and there is 50% and 41% lower participation for the two control groups, respectively. The lower participation can be explained in part due to students dropping the course during the semester. For example, the enrollment at the end of the spring 2011 semester was 25 students, and for the spring 2012 semester, enrollment included 18 students. There was also the issue of different students taking the pretest and posttest. For example, in spring 2011, 3 students took the pretest but did not take the posttest, and 2 students took the posttest but not the pretest. For spring 2012, 1 student took the pretest but did not take the posttest, and 7 students took the posttest but not the pretest. An analysis of the posttest results of students in the spring 2012 who took the posttest but not the pretest would not have significantly changed the posttest scores.

Results of the pre/posttest shown in Table III suggest that the control and treatment groups were taken from two different populations. This is mainly due to the mean pretest scores for the control groups being higher than the treatment groups. The students selected for the study were assigned based on the semester in which they took the SE class. FIU is a commuter school, and with a large proportion of nontraditional students, most FIU students take classes year-round. There is no difference in the student body in the fall, spring, and summer sessions.

The anomaly of the scores in the pretest for the control groups was that in spring 2012, two students in the SE class took the undergraduate software testing class in fall 2011, and one student works in industry as a software tester. There was also one student in the summer 2011 class who took the undergraduate software testing class in fall 2010. If the students who previously took the software testing class were removed from the raw data, then the mean scores for the pretest in Tables III through VI would be closer together. For example, the mean scores for the pretest control and treatment groups in Table III would only differ by 0.8.

Three different instructors were used during the study—one each for the spring 2011 and 2012 control groups, and one for the summer 2011 and 2012 treatment groups. The instructors used in the study were based on the departmental assignment. The major threat to validity of the instructor assignment was that the instructor assigned to the

treatment groups was an experienced software testing instructor and tended to place more emphasis on verification and validation in the SE course. To mitigate this threat, the instructors used the same class material and structure for the project deliverables.

5. RELATED WORK

In this section, we present the related work on the approaches used to introduce testing into programming courses and compare several of the online resources that contain software testing learning materials to WReSTT. We also compare our work to systems that use collaborative learning in CS/IT.

5.1. Testing in Programming Classes

Integrating testing into courses. There have been several approaches used to integrate testing into courses at various stages in the CS/IT curriculum. Goldwasser [2002] integrated black box testing into a programming course by encouraging students to submit a test set with each programming assignment. Barbosa et al. [2003] describe an approach to introduce testing earlier in the software development process that focuses on students recognizing the importance of the testing activity and motivating students to use testing ideas in their projects. Frezza [2002] describes an approach that integrates testing into an introductory software design course. This work was motivated by the fact that integrating testing and design topics allows students to experience different design concepts and reinforces the importance of testing. Janzen and Saiedian [2006] present TDL as a pedagogical tool that can be incorporated into multiple levels of the CS and SE curricula. TDL is a pedagogical approach used to teach TDD and has the following objectives: (1) teach testing for free, (2) encourage the use of TDD, (3) improve student programming ability, and (4) improve the quality of the design and correctness of programs.

Unlike the approaches by Barbosa et al. [2003], Frezza [2002], and Janzen and Saiedian [2006], our approach uses a *minimally disruptive* approach from the instructor's point of view. That is, there is no need for major changes to the syllabus or restructuring of the course to integrate testing. The onus is on the students to gain the testing knowledge from WReSTT. Similar to the approach of Goldwasser [2002], we advocate that students submit the test set with the programs that they are testing, as well as the code coverage results for the test set. We believe that WReSTT can be integrated effectively in the the preceding approaches and can provide supplemental and engaging learning opportunities for students.

Encouraging testing using grading tools. One of the most successful projects on using grading tools to assist instructors and students is the Web-Based Center for Automated Testing (Web-CAT) developed by Edwards [2003]. Several researchers who included testing early in the curriculum [Goldwasser 2002; Edwards 2003; Jones 2000] identified the need for tools that support automatic grading of student assignments. Software testing can play an important role in learning to develop software, but only when students are provided with a special environment that provides frequent feedback on their performance in forming hypotheses and verifying them. Although Web-CAT is very good for automated assessment of student programs, it does not claim to provide tutorials on (1) how to generate test cases, (2) how code coverage works, or (3) using functional testing tools (see Table I). In our opinion, WReSTT can complement Web-CAT by providing students with tutorials to help them better understand the testing process.

5.2. Online Testing Resources

There are several online resources that provide access to learning materials on software testing to the academic community. During the past decade, Kaner et al. [2001]

developed the Center for Software Testing Education and Research (CSTER), which has a large repository of content on software testing including video lectures, practice quizzes, drills, and other assessment materials. Williams [2010] developed a module on software testing (Module 7) in OpenSeminar, a Web-based open courseware platform that contains examples, lab exercises, lectures (slides), and readings.

Cassel et al. [2012] coordinate the activities of the Ensemble Computing Portal, which is a U.S. National Science Foundation (NSF)-sponsored National Science Digital Library (NSDL) Pathways project for computing education materials. Ensemble provides access to other repositories containing software testing materials such as SWENET [Lutz et al. 2010] and CITIDEL [CITIDEL Team 2010]. Garousi [2010] describes a Web repository developed by the Software Quality Engineering Research Group (SoftQual) that contains lab exercises focused on tools that support testing used at a cross section of SE programs in North America. These tools include Bugzilla, JUnit, CodeCover, CodeLipse, Rational Functional Tester, and MuClipse [Garousi 2010; Software Quality Engineering Research Group (SoftQual) 2012].

Few, if any, of the aforementioned online software testing repositories provide facilities for instructors to monitor how students in their classes use the provided resources, such as tutorials and quizzes. In addition, none of them provide students with a collaborative learning experience where students need to work together in teams to accomplish a task and earn virtual points based on their performance. The repositories mentioned earlier are more static when compared to WReSTT, which is dynamic with respect to student and instructor interaction. Note, however, that many of these repositories provide valuable resources, such as tutorials (using different formats), quizzes, and labs on a variety of testing topics that can be accessed by WReSTT.

5.3. Collaborative Learning

Talon et al. [2009] describe their approach to software testing using a platform that supports collaborative activities. The authors applied the MAETIC pedagogical method, which organizes project-based pedagogy. To support the MAETIC approach, the Cooperative Layer Supporting Distributed Activities (CooLDA) platform is used to support the collaborative activities. The basic idea of the approach is to use CooLDA to keep track of the tools associated with the specific activities of a software testing project. The testing activities for the project, the source code to be tested, software requirements, and functional specifications are all managed by CooLDA.

The popularity and acceptance of e-learning, as well as the growing availability of cyberinfrastructure Ramírez and Fox [2011], has driven the need for tools to support collaboration learning. Li et al. [2008] survey several of the technology issues, including some of the tools, needed to support distributive and collaborative learning. Several of the issues addressed by the authors include standards used to create learning materials, hosting of e-learning systems as the content and user base continue to grow, support for content delivery using different multimedia formats, and support for synchronous and asynchronous learning modes. These tools included Moodle, ANGEL Learning Management Suite (LMS), and ATutor.

WReSTT is different from CooLDA [Talon et al. 2009] because it does not provide support for project management activities or keep track of the tools students used to test their programs. The current version of WReSTT (V2) incorporates many of the ideas presented by Talon et al. [2009] and addresses several of the issues discussed by Li et al. [2008]. These ideas include keeping track of student activities as they access the various learning materials, using the Web as a platform to host learning materials and as a collaborative platform, and providing learning materials in different formats. In the future, we plan to transform existing materials and create new learning materials

using a standard format (e.g., learning objects) so that the materials can be used by other systems.

6. CONCLUSIONS

In this article, we described a minimally disruptive approach to integrating software testing into SE courses. The approach presented is not a replacement for existing modules taught in SE courses but is meant to supplement them by providing a collaborative learning environment that engages students and encourages software testing learning activities. These learning materials are hosted in a Web-based repository referred to as WReSTT. To determine the impact that using WReSTT had on students' ability to learn software testing concepts and testing tools, we conducted studies involving students in SE classes over a period of 2 years.

The studies involved two versions of WReSTT, the first version (V1) with only tutorials on testing tools, and the second version (V2) containing the tutorials from V1 with added collaborative learning and social engagement features. The results of the studies showed that (1) WReSTT as a teaching tool for students in SE courses can improve their understanding of software testing techniques, (2) WReSTT as a teaching tool for students in SE courses can improve their knowledge and use of testing tools, and (3) the changes made from WReSTT V1 to V2 have improved students' understanding and use of software testing concepts and tools. Result (3) is particularly important because of the use of the collaborative learning features and improved usability in WReSTT V2.

As a result of the studies performed in this project, several areas of future research have been identified. These research directions would require extending the functionality of WReSTT and using more direct measures in the studies. We plan to extend the learning materials in WReSTT and restructure them so that the tutorials on testing concepts and tools are organized into learning objects. Learning objects would provide a more structured approach to students who plan to use WReSTT as an independent learning resource. Using more direct measures, such as observing students using the testing tools, would allow for a more in-depth and detailed analysis on the various components that are part of learning software testing, such as students' ability to use black box and/or white box testing techniques. In addition, we can perform more focused analyses on detailed usage information of WReSTT, student team composition, and social engagement levels.

APPENDIX

A. PRETEST/POSTTEST

This test will NOT impact your course grade.
ID Number (DO NOT use your name):

1. What is the main objective of program testing?
[*1 point*]
Ans: To find bugs/errors/faults in a program or any sentence that conveys this idea.

2. (a) Identify all the testing techniques you have used to create test cases:
[*1 point for each correct answer*]
Ans: (a) Equivalence partitioning, boundary analysis, random selection, state-based, among others.
(b) Write test cases to test the code provided below. Identify the testing technique used to generate the test case.
//Method to withdraw money from a bank account

//The following variables are defined as follows: requiredMin = 50.0 and balance = 100;

```
public void withdraw (double amount) {
   if ((balance - amount) < requiredMin)
      System.out.println(''Insufficient funds'');
   else
      balance = balance - amount;
}
```

Input Value	Expected Output Values			Testing Technique
amount	amount	balance	requiredMin	

[*1 point for each complete test case*]

Ans: (b)(i) Numeric values for all the attributes in the table were given.

[*1 point for each correct test case*]

(ii) Values were checked for correctness based on the semantics of the method.

[*1 point for each correct testing technique*]

(iii) Any of the answers in 2(a).

[*1 point for matching correct testing technique to test case*]

(iv) The input value provided a guide to whether or not a particular testing technique was used, e.g., amount = 49.9 would signify boundary analysis, amount = –1 equivalence partitioning

3. Have you ever used tools to support testing of programs? Circle your answer:
   Yes No
   [*1 point for Yes*]

4. If you have answered "Yes" to Question (3), answer the following questions:

   a. List the names of the tools you have used:
   [*1 point for each correct tool listed*]
   Ans: JUnit, PhPUnit, OCUnit, GHUnit, Visual Studio Unit Test, IBM Rational Functional Tester (RTF), SWAT, Cobertura, EclEmma, CoverStory

   b. List one tool in each of the following categories that you have used and indicate your level of proficiency corresponding to each tool on a scale of 1–5 with *1 = barely competent* and *5 = extremely proficient*:
   [*1 point for each correct tool listed per category; points = proficiency score, if correct tool listed*]

	Category	*Tool(s)*	*Proficiency*
i	Unit Testing	***JUnit, PhPUnit, OCUnit, GHUnit, Visual Studio Unit Test***	
ii	Functional Testing	***IBM Rational Functional Tester (RFT), SWAT***	
iii	Code Coverage	***Cobertura, EclEmma, CoverStory***	

5. Do you know of any online resources that provide information on software testing?
   Yes No
   [*1 point for Yes*]

6. If you have answered "Yes" to Question (5), answer the following questions:

   a. State the names of online resources.
      [*1 point for stating a correct online resource*]
      Ans: WReSTT, Apple Developer, Wikipedia, YouTube, Udacity, among others

   b. State the type of materials (notes, lab exercises, tutorials, etc.) found at each resource listed above.
      [*1 point for stating a valid material type*]
      Ans: Tutorials, Video Tutorials, Tool Documentation, Quizzes, Notes on Testing, among others

7. How beneficial do you think it is to use tools to support testing of your programming assignments? Use a scale of 1–5 with *1 = Not beneficial* and *5 = Extremely beneficial*

8. If you answered 2 or above to Question (7), state one reason for your answer.

B. STUDENT SURVEY

This survey will NOT impact your course grade.

Panther ID Number (DO NOT use your name):

1. Have you ever used a learning resource other than WReSTT to learn about testing concepts or testing tools? Please circle your answer: Yes No

You have been exposed to two versions of WReSTT, version 1 (version with no points) and version 2 (version with points). Please use the following scale and complete questions 2–15: *1 = Strongly Disagree, 2 = Disagree, 3 = Neither Agree nor Disagree, 4 = Agree, 5 = Strongly Agree, NA = Not Applicable*. Please circle the appropriate answer.

.

	Overall Reaction to Web Sites	Rating V1	Rating V2
2	Overall, I am satisfied with how easy it is to use the Web site.		
3	It is simple to use the Web site.		
4	I feel comfortable using the Web site.		
5	It was easy to learn to use the Web site.		
6	I believe I became productive quickly using the Web site.		
7	The information (such as online help, on-page messages, and other documentation) provided with the Web site is clear.		
8	It is easy to find the information I need.		
9	The information is effective in helping me complete the tasks and scenarios.		
10	The interface of the Web site is pleasant.		
11	I like using the interface of this Web site.		
12	The Web site has all the functions and capabilities I expect it to have.		
13	I believe that the Web site helped me earn a better grade.		
14	I would recommend the Web site to fellow students.		
15	Overall, I am satisfied with the Web site.		

Please use the following scale and complete questions 16–21: *1 = Strongly Disagree, 2 = Disagree, 3 = Neither Agree nor Disagree, 4 = Agree, 5 = Strongly Agree, NA = Not Applicable.*

	Testing-Related Questions	Rating
16	The tutorials in WReSTT helped me to better understand testing concepts.	
17	The tutorials in WReSTT helped me to better understand how to use unit testing tools.	
18	The tutorials in WReSTT helped me to better understand how to use code coverage testing tools.	
19	The tutorials in WReSTT helped me to better understand how to use functional testing tools.	
20	The number of tutorials in WReSTT is adequate.	
21	I would have used testing tools in my project if WReSTT did not exist.	

The following questions are related to WReSTT Version 2 only. Please use the following scale and complete questions 22–26: *1 = Strongly Disagree, 2 = Disagree, 3 = Neither Agree nor Disagree, 4 = Agree, 5 = Strongly Agree, NA = Not Applicable.*

	Collaborative Learning–Related Questions	Rating
22	The use of virtual points in WReSTT V2 encouraged me to visit the Web site and complete the tasks.	
23	The use of virtual points in WReSTT V2 encouraged my team to visit the Web site and complete the tasks.	
24	The event stream showing the activities of the other members in the class encouraged me to complete my tasks in WReSTT V2.	
25	The event stream showing the activities of the other members in the class encouraged my team to complete my tasks in WReSTT V2.	
26	Our team devised a plan to get the maximum number of points in WReSTT V2.	

Open ended questions:

27. Compare your experience using WReSTT version 1 and WReSTT version 2:
    (a) What did you like more in WReSTT version 1 than WReSTT version 2?
    (b) What did you like more in WReSTT version 2 than WReSTT version 1?

28. List any other features you would like to see in WReSTT version 2.

29. State your team number and the number of virtual points you have.
    Team number: Virtual Points: (as a percentage)

30. Any other comments:

ACKNOWLEDGMENT

We would like to thank Dionny Santiago and Yesenia Sosa for their work on developing WReSTT V2. We would also like to acknowledge Frank Hernandez and Xabriel J Collazo-Mojica for assisting with the studies in the spring of 2011 and 2012, respectively.

REFERENCES

ACM/IEEE-CS Interim Review Task Force. 2008. Computer Science Curriculum 2008: An Interim Revision of CS 2001. Retrieved August 7, 2014, from http://www.acm.org/education/curricula/ComputerScience2008.pdf.

Paul Ammann and Jeff Offutt. 2008. *Introduction to Software Testing.* Cambridge University Press, New York, NY.

Tara Astigarraga, Eli M. Dow, Christina Lara, Richard Prewitt, and Maria R. Ward. 2010. The emerging role of software testing in curricula. In *Transforming Engineering Education: Creating Interdisciplinary Skills for Complex Global Environments, 2010 IEEE*. IEEE, Los Alamitos, CA, 1–26. DOI: http://dx.doi.org/10.1109/TEE.2010.5508833

Ellen Francine Barbosa, José Carlos Maldonado, Richard LeBlanc, and Mark Guzdial. 2003. Introducing testing practices into objects and design course. In *Proceedings of the 16th Conference on Software Engineering Education and Training*. IEEE, Los Alamitos, CA, 279–286.

Victor R. Basili, Richard W. Selby, and David H. Hutchens. 1986. Experimentation in software engineering. *IEEE Transactions on Software Engineering* SE-12, 7, 733–743. DOI: http://dx.doi.org/10.1109/TSE.1986.6312975

Sebastian Bergmann. 2013. PHPUnit. Retrieved August 7, 2014, from https://github.com/sebastianbergmann/phpunit/.

Robert V. Binder. 1999. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison Wesley Longman, Boston, MA.

Pierre Bourque and Robert Dupuis. 2004. *Guide to the Software Engineering Body of Knowledge 2004 Version*. IEEE, Los Alamitos, CA.

Bernd Bruegge and Allen H. Dutoit. 2009. *Object-Oriented Software Engineering Using UML, Patterns, and Java* (3rd ed.). Prentice Hall, Upper Saddle River, NJ.

Lilian Cassel, Lois Delcambre, Edward Fox, and Richard Furuta. 2012. Ensemble: Computing Portal Connecting Computing Educators. Retrieved August 7, 2014, from http://www.computingportal.org/.

Yaofei Chen, Rose Dios, Ali Mili, Lan Wu, and Kefei Wang. 2005. An empirical study of programming language trends. *IEEE Software* 22, 3, 72–78.

CITIDEL Team. 2010. CITIDEL: Computing and Information Technology Interactive Digital Educational Library. Retrieved August 7, 2014, from http://www.citidel.org/.

Mike Clark. 2012. JDepend. Retrieved August 7, 2014, from http://clarkware.com/software/JDepend.html.

Peter J. Clarke, Andrew A. Allen, Tariq M. King, Edward L. Jones, and Prathiba Natesan. 2010. Using a Web-based repository to integrate testing tools into programming courses. In *Proceedings of the ACM OOPSLA 2010 Companion (SPLASH'10)*. ACM, New York, NY, 193–200.

Peter J. Clarke, Jairo Pava, Debra Davis, and Tariq M. King. 2012. Using WReSTT in SE courses: An empirical study. In *Proceedings of the 43rd SIGCSE Conference*. ACM, New York, NY, 307–312.

Peter J. Clarke, Jairo Pava, Yali Wu, and Tariq M. King. 2011. Collaborative Web-based learning of testing tools in SE courses. In *Proceedings of the 42nd SIGCSE Conference*. ACM, New York, NY, 147–152.

CNN. 2012. Is Knight's $440 million glitch the costliest computer bug ever? (2012). *CNN Money*, August 9. Retrieved August 7, 2014, from http://money.cnn.com/2012/08/09/technology/knight-expensive-computer-bug/index.html.

CNSS. 2005. *Software 2015: A National Software Strategy to Ensure U.S. Security and Competitiveness*. Technical Report. Center for National Software Studies, Upper Marlboro, MD.

Cobertura Team. 2012. Cobertura. Retrieved August 7, 2014, from http://cobertura.sourceforge.net/.

CoverStory Team. 2013. CoverStory. Retrieved August 7, 2014, from http://code.google.com/p/coverstory/.

Tony Cowling. 2012. Stages in teaching software testing. In *Proceedings of the 2012 International Conference on Software Engineering (ICSE'12)*. IEEE, Los Alamitos, CA, 1185–1194. Available at http://dl.acm.org/citation.cfm?id=2337223.2337379.

David C. Crowther and Peter J. Clarke. 2005. Examining software testing tools. *Dr. Dobbs Journal* 373, 1, 26–33.

Chetan Desai, David S. Janzen, and John Clements. 2009. Implications of integrating test-driven development into CS1/CS2 curricula. *ACM SIGCSE Bulletin* 41, 1, 148–152.

Adrián Domínguez, Joseba Saenz de Navarrete, Luis de Marcos, Luis Fernández-Sanz, Carmen Pagés, and José-Javier Martínez-Herráiz. 2013. Gamifying learning experiences: Practical implications and outcomes. *Computers and Education* 63, 380–392.

Drupal Community. 2012. Drupal. Retrieved August 7, 2014, from http://drupal.org/.

Thomas Dvornik, David S. Janzen, John Clements, and Olga Dekhtyar. 2011. Supporting introductory test-driven labs with WebIDE. In *Proceedings of the 24th IEEE-CS Software Engineering Education and Training Conference (CSEET'11)*. IEEE, Los Alamitos, CA, 51–60.

Stephen H. Edwards. 2003. Rethinking computer science education from a test-first perspective. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'03)*. ACM, New York, NY, 148–155.

Facebook Team. 2012. Facebook. Retrieved August 7, 2014, from http://www.facebook.com/.

Michael Feathers. 2012. CppUnit. Retrieved August 7, 2014, from http://sourceforge.net/projects/cppunit/.

Stephen Frezza. 2002. Integrating testing and design methods for undergraduates: Teaching software testing in the context of software design. In *Proceedings of the 32nd Annual Frontiers in Education, 2002 (FIE'02), Vol 3*. IEEE, Los Alamitos, CA, SIG-1–SIG-4.

Michael P. Gallaher and Brian M. Kropp. 2011. The Economic Impacts of Inadequate Infrastructure for Software Testing. Retrieved August 7, 2014, from http://www.rti.org/pubs/software_testing.pdf.

Erich Gamma and Kent Beck. 2012. JUnit. Retrieved August 7, 2014, from http://www.junit.org/.

Vahid Garousi. 2010. An open modern software testing laboratory courseware—an experience report. In *Proceedings of the 23rd IEEE-CS Software Engineering Education and Training Conference (CSEET'10)*. IEEE, Los Alamitos, CA, 177–184.

Michael H. Goldwasser. 2002. A gimmick to integrate software testing throughout the curriculum. In *Proceedings of the 33rd SIGCSE Conference*. ACM, New York, NY, 271–275.

Starr R. Hiltz. 1998. Collaborative learning in asynchronous learning networks: Building learning communities. (1998). In *Proceedings of WEB98*.

Marc R. Hoffmann. 2012. EclEmma. Retrieved August 8, 2014, from http://www.eclemma.org/.

IBM. 2012. Rational Functional Tester. Retrieved August 7, 2014, from http://www-01.ibm.com/software/awdtools/tester/functional/.

David S. Janzen and Hossein Saiedian. 2006. Test-driven learning: Intrinsic integration of testing into the CS/SE curriculum. *ACM SIGCSE Bulletin* 38, 1, 254–258. DOI:http://dx.doi.org/10.1145/1124706.1121419

Edward L. Jones. 2000. Software testing in the computer science curriculum—a holistic approach. In *Proceedings of the Australasian Conference on Computing Education (ACSE'00)*. ACM, New York, NY, 153–157.

Cem Kaner, James Bach, and Bret Pettichord. 2001. *Lessons Learned in Software Testing*. John Wiley & Sons, New York, NY.

Timothy C. Lethbridge, Jorge Diaz-Herrera, Richard J. LeBlanc Jr., and J. Barrie Thompson. 2007. Improving software practice through education: Challenges and future trends. In *Proceedings of the 2007 Future of Software Engineering (FOSE'07)*. IEEE, Los Alamitos, CA, 12–28.

Cen Li, Zhijiang Dong, Roland H. Untch, and Michael Chasteen. 2013. Engaging computer science students through gamification in an online social network based collaborative learning environment. *International Journal of Information and Education Technology* 3, 1, 72–77.

Qing Li, Rynson W. H. Lau, Timothy K. Shih, and Frederick W. B. Li. 2008. Technology supports for distributed and collaborative learning over the Internet. *Transactions on Internet Technology* 8, 2, Article No. 5. DOI:http://dx.doi.org/10.1145/1323651.1323656

Michael J. Lutz, W. Michael McCracken, Susan Mengel, Mark Sebern, Greg W. Hislop, and Thomas B. Hilburn. 2010. SWENET—SEEK Category: Software Verification and Validation (VAV). Retrieved August 7, 2014, from http://www.swenet.org/browseModules.aspx?categoryID=11.

Thomas W. Malone. 1980. What makes things fun to learn? Heuristics for designing instructional computer games. In *Proceedings of the 3rd ACM SIGSMALL Symposium and the 1st SIGPC Symposium on Small Systems (SIGSMALL'80)*. ACM, New York, NY, 162–169. DOI:http://dx.doi.org/10.1145/800088.802839.

Aditya P. Mathur. 2008. *Foundations of Software Testing.* Pearson Education, Delhi, India.

Microsoft Corporation. 2013. Using Testing Tools in Visual Studio Professional Edition. Retrieved August 7, 2014, from http://msdn.microsoft.com/en-us/library/bb385902(v=VS.90).aspx.

Glenford J. Myers. 2004. *Art of Software Testing* (2nd ed.). John Wiley & Sons, New York, NY.

John Neter, William Wasserman, and Michael H. Kutner. 1990. *Applied Linear Statistical Models*. Irwin Press, Boston, MA.

William Perry. 2006. *Effective Methods for Software Testing* (3rd ed.). John Wiley & Sons, New York, NY.

Shari Lawrence Pfleeger and Joanne M. Atlee. 2009. *Software Engineering: Theory and Practice* (4th ed.). Pearson Education, Cranbury Township, NJ.

Alex Ramírez and Geoffrey C. Fox. 2011. A Report of the National Science Foundation Advisory Committee for Cyberinfrastructure Task Force on Cyberlearning and Workforce Development. Retrieved August 7, 2014, from http://www.nsf.gov/cise/aci/taskforces/TaskForceReport_Learning.pdf.

Sen:te. 2013. Moving OCUnit tests between Logic and Application tests. Retrieved August 7, 2014, from http://www.sente.ch/?p=276&lang=en.

Terry Shepard, Margaret Lamb, and Diane Kelly. 2001. More testing should be taught. *Communications of the ACM* 44, 6, 103–108. DOI:http://dx.doi.org/10.1145/376134.376180

Charlie Y. Shim, Mina Choi, and Jung Yeop Kim. 2009. Promoting collaborative learning in software engineering by adapting the PBL strategy. In *Proceedings of the WASET International Conference on Computer and Information Technology (ICCIT '09)*. IEEELos Alamitos, CA, 1167–1170.

Sidney Siegel and N. John Castellan. 1988. *Nonparametric Statistics for the Behavioral Sciences* (2nd ed.). McGraw-Hill, New York, NY.

Barbara Leigh Smith and Jean T. MacGregor. 1992. What is collaborative learning? In *Collaborative Learning: A Sourcebook for Higher Education*, A. S. Goodsell, M. R. Maher, and V. Tinto (Eds.). National Center on Postsecondary Teaching, Learning, and Assessment, University Park, PA.

Software Quality Engineering Research Group (SoftQual). 2012. SoftQual Repository of Software Testing Laboratory Courseware. Retrieved August 7, 2014, from http://www.softqual.ucalgary.ca/projects/testing_labs/.

Ian Sommerville. 2004. *Software Engineering* (7th ed.). Pearson Addison Wesley, Boston, MA.

Bénédicte Talon, Dominique Leclet, Arnaud Lewandowski, and Grégory Bourguin. 2009. Learning software testing using a collaborative activities oriented platform. In *Proceedings of the 2009 9th IEEE International Conference on Advanced Learning Technologies (ICALT'09)*. IEEE, Los Alamitos, CA, 443–445. DOI:http://dx.doi.org/10.1109/ICALT.2009.61

William Trochim. 2001. *The Research Methods Knowledge Base* (2nd ed.). Atomic Dog Publishing, Cincinnati, OH.

Tom Tullis and Bill Albert. 2008. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Elsevier: Morgan Kaufmann, San Francisco, CA.

Ultimate Software. 2012. Simple Web Automation Toolkit. Retrieved August 7, 2014, from http://sourceforge.net/projects/ulti-swat/.

Wikipedia. 2012. List of Software Bugs. Retrieved August 7, 2014, from http://en.wikipedia.org/wiki/List_of_software_bugs.

Laurie Williams. 2010. Software Engineering: Testing. *OpenSeminar*. Retrieved August 7, 2014, from http://openseminar.org/se/modules/7/index/screen.do.

Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2000. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA.

WReSTT Team. 2012. WReSTT: Web-based Repository of Software Testing Tools. Retrieved August 7, 2014, from http://wrestt.cis.fiu.edu/.

Received December 2012; revised April 2014; accepted April 2014