Using a Web-Based Repository to Integrate Testing Tools into Programming Courses

Peter J. Clarke, Andrew A. Allen

School of Computing and Info. Sciences
Florida International University
Miami, FL 33199, USA
{clarkep, aalle004}@cis.fiu.edu

Tariq M. King

Dept. of Computer Science
North Dakota State University
Fargo, ND 58108, USA
tariq.king@ndsu.edu

Edward L. Jones

Dept. of Computer and Info. Sciences
Florida A&M University
Tallahassee, FL 32307, USA
ejones@cis.famu.edu

Prathiba Natesan

Department of Educational Psychology
University of North Texas
Denton, TX 76203, USA
prathiba.natesan@unt.edu

Abstract

Improving the quality of software developed in the 21st century is one of the major challenges in the software industry. Addressing this problem will require that academic institutions play a key role in training developers to develop high quality software. Unfortunately, students and instructors continue to be frustrated by the lack of support provided when selecting appropriate testing tools and program analyzers to verify programs under development.

In the paper we present an approach that integrates the use of software testing tools into programming and software engineering courses. The approach consists of three phases, developing an online repository with learning resources, training instructors in the area of testing techniques and tools, and integrating the use of testing tools into various programming courses. We also present the results of the first instructors' workshop and for studies on integrating testing tools into two courses, CS2 and Software Engineering (SE).

Categories and Subject Descriptors   K.3.2 [*Computer and Information Science Education*]: Miscellaneous

General Terms   Experimentation

Keywords   Software Testing, Unit Testing, Programming Courses, Computer Science Education

1. Introduction

The size and complexity of software systems continue to grow as software becomes more pervasive and ubiquitous. Ensuring the quality of these software systems in the 21st Century will require changes to the development strategies and improvement to the pedagogy used to teach these strategies in academia. Testing continues to be the primary technique used to ensure the development of high quality software, but recent studies [2, 30] indicate that major improvements in software testing are needed. Any comprehensive approach to improving the quality of software systems developed in the future requires that academic institutions play a vital role in training students how to test software, and making them aware of the tools available to support software testing. Unfortunately, students and instructors continue to be frustrated by the lack of support provided when selecting appropriate testing tools and program analyzers to verify programs under development.

Research data on the use of software testing tools by students and instructors to support pedagogy are scarce. During the last decade several researchers [14, 16, 19–21] have indicated that little or no coverage is given to software design and testing techniques in many academic CS programs. One of the first Computer Science (CS) and Information Technology (IT) courses that students encounter in many programs in the nation is "CS1 - Introduction to Programming". It is in CS1 that students should be exposed to the tools that have the potential to improve the quality of the code they write. Exposing students to testing techniques and tools during courses early in the CS/IT curricula would allow them to gain the necessary practice and experience required to produce high quality software. There are also re-

ports that little attention is given to software testing even in software engineering (SE) courses. One aspect not fully addressed in the research papers on integrating testing into CS/IT courses is the preparation required to introduce programming course instructors to the area of software testing. To support the aforementioned claim, Lethbridge et al. [24] raise the broader question of how many instructors teaching core SE courses have a deep background in the field.

In this paper we present an approach that supports the integration of testing into programming courses supported by a web-based repository of testing tools. The approach consists of three phases: (1) developing an online portal of learning resources that supports pedagogy in the area of software testing; (2) holding a series of annual workshops for instructors to introduce them to software testing and the learning resources available through the online portal; and (3) integrating the use of testing tools into programming and SE courses. We report on the phases of the project mentioned above identifying the results obtained from the studies that have been conducted to date.

The remainder of the paper is organized as follows: in Section 2 we describe techniques in the literature that integrates testing into programming courses. Section 3 presents our approach and Section 4 describes the results of our first instructors' workshop and studies conducted in CS2 and SE courses. Section 5 describes the related work and we conclude in Section 6.

2. Testing in Programming Courses

Members of the academic community and software industry have expressed interest in integrating testing into the curricula of programming courses. The most recent approach being employed by instructors is Test Driven Learning (TDL), or teaching Test Driven Development (TDD). Such approaches involve using automated testing tools to motivate students to learn about software testing, while improving their testing skills and ability to develop quality software. In this section we describe some of the approaches being used by instructors to integrate testing into programming courses, and summarize the results of applying these approaches as reported in the literature.

2.1 Approaches

Goldwasser [16] proposed a technique that required minor modifications to the course structure to include the submission of test sets with the source code for assignments. The source code of each assignment is then executed on each test set submitted. This approach uses mainly black-box testing [27]. Goldwasser refers to this approach as a "little gimmick" and states that the approach can be applied to most programming courses at all level of the curriculum. A significant benefit for instructors cited in this approach is its amenability to existing programming assignments. This approach requires unambiguous specifications and automated

grading tools to support medium to large classes [16]. As such an instructor would need to learn how to create such assignments and use these tools.

Edwards [7, 8] observed that most beginner students use the trial-and-error approach to program correctness. However, even though natural and sufficient for simple programs, this approach does not lead to the development of higher order problem solving skills needed to develop more complex software. Students need practice applying the scientific method: observing the behavior of software, hypothesizing cause-effect relationships in the software, and experimenting to verify hypotheses. 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 forming hypotheses and verifying them. Web-CAT, developed by Edwards, receives a student's program code and test set, and provides as feedback an overall score (grade) and three performance measures: (1) code correctness based on the number of student-supplied test cases that passed; (2) code test completeness based on code coverage; and (3) problem test completeness and validity based on a teacher-provided test set.

Janzen et al. [20] present test-driven learning (TDL) as a pedagogical tool and discuss the incorporation of TDL into multiple levels of the CS and SE curricula. They propose that TDL can be applied as early as the first day of the first programming course, but that it should not compete with other approaches in introductory courses. Instead, TDL should integrate well with other programming-first approaches such as imperative-first, objects-first, functional-first, event-driven, among others. An experiment was conducted in two CS1 sections, taught by the same instructor, to compare the scores between TDL students and non-TDL students.

Elbaum et al. [11] state that it continues to be difficult to integrate testing into early programming courses due to the lack of appropriate courseware materials that can be directly used by instructors. Therefore in an effort to promote early integration of software testing into early programming courses, the authors developed a hands-on web-based tutorial named *Bug Hunt*. The features of Bug Hunt provide both instructors and students with the ability to: (1) practice the fundamentals of testing while providing students with feedback; (2) review the material in the tutorial at their pace; (3) configure the tutorial to accommodate the instructor's requirements; and (4) automatically assess the students' performance by the instructor.

Schaub [31] describes an instructor's experience when introducing testing into a CS1 course that focuses on web application development. The author combines the use of a web API, an appropriate development environment, and a TDD methodology to emphasize design and testing. TDD was employed to get students to focus on the design of the

core application API, while writing automated unit tests before implementing the application's functionality. For assignments, students were provided with a specification of the classes and methods to be developed, together with an initial unit test set that must pass for the solution to be considered acceptable. To reduce the effort needed for students to adopt unit testing techniques in CS1, the approach used by Schaub [31] did not require students to learn xUnit-style frameworks [33] in CS1 but instead write stand-alone unit tests as a console application.

The work by Desai et. al [5] demonstrate how TDD can be integrated into CS1/CS2 course curricula. The approach attempts to introduce testing without burdening the students by giving them full JUnit [15] test suites for projects and labs early. Initially, JUnit tests are supplied for a Java class similar to one the students would have to test, e.g., test cases were supplied for a *Triangle* class, and the students had to write the test cases for a *Rectangle* class. However, in subsequent projects students were expected to write all the tests themselves without the aid of test examples. Students were also taught the value of reusable automated unit tests as projects built upon previous ones. Therefore, some written tests did not have to change but just re-run to ensure that changes did not break any of the previously tested functionality. Two controlled experiments were conducted to evaluate the introduction of TDD into a CS1/CS2 course.

2.2 Summary of Results

Some observations from the aforementioned works on integrating testing into programming courses can be summarized as follows:

- Careful structuring of assignments is key in getting students to adopt the test-driven approach naturally.

- The use of adequate automatic grading tools is essential for: (1) reducing the instructor's course load, and (2) providing students with helpful feedback on the adequacy of the testing done in their assignments.

- Simply re-writing course materials to incorporate TDD, even though effective, is not ideal. Re-ordering and re-emphasis of topics is recommended.

- It can be difficult to determine what kind of curriculum changes are required, how drastically projects must change and what will be the effect on the students.

- Aside from the one-time setup cost, instructor effort is not necessarily increased with the introduction of TDL, and in some cases may even decrease thanks to the opportunity of automated grading.

3. Using WReSTT to Support Pedagogy

There have been several approaches used to integrate testing into programming courses as described in the previous section. Although these approaches expect instructors to participate in this process, there needs to be more work done

to assist instructors in gaining the necessary knowledge on testing to support the integration. In addition to this knowledge transfer, it is important to complement this integration with appropriate tools, course materials and other resources. In this section we describe an approach consisting of three phases that integrates testing into programming courses.

3.1 Web-Based Repository

Phase one of our approach consists of developing an online portal of learning resources that supports pedagogy in the area of software testing. The main objective of the online portal is to increase the number of users at academic institutions that currently have access to vetted learning materials, including tutorials on software testing tools, that support the integration of testing into programming courses. We have created such a repository known as *Web-based Repository of Software Testing Tools (WReSTT)*[1] [35] that contains tutorials on software testing tools and links to other materials on software testing. WReSTT currently contains learning materials for the following tools:

- *Cobertura* - a free Java tool that calculates the percentage of code accessed by tests [3].

- *CppUnit* - a C++ unit testing framework [13].

- *EclEmma* - a free Java code coverage tool for Eclipse [17].

- *JDepend* - a tool that traverses Java class file directories and generates design quality metrics for each Java package [1].

- *JUnit* - a unit testing framework for the Java programming language [15].

- *SWAT* - the Simple Web Automation Toolkit (SWAT) is a library written in C# designed to provide an interface to interact with several different web browsers [32].

- *Rational Functional Tester* - an automated functional and regression testing tool [18].

Figure 1 shows the web page containing the learning materials for JUnit [15]. The top of the page contains links to the registration page, the forums, events related to the project, sponsors of the project, links to other learning materials, and the contact information for the page. The left side of the page is the sidebar that allows users to navigate around the portal and a list of participating institutions. The center page contains the content for JUnit, including a link to the official JUnit web site, and video tutorials on setting up JUnit in Eclipse and creating test cases for a simple example program. The right side of the page contains the links to other testing resources and recent events.

WReSTT was developed using Drupal [6], a content management system, and uses a four-tier architecture. WReSTT was designed to allow access to four types of users: develop-

---

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

Figure 1. Web page in WReSTT showing the links and some of the tutorials for JUnit.

ers, moderators, instructors and students. Each type of user has access to a different set of facilities in WReSTT, e.g., instructors can monitor how frequently students in their classes access the tutorials for a specific tool. Data is maintained for each type of user thereby allowing the WReSTT team to monitor the use of the repository.

3.2 Instructors Workshop

Phase two of our approach consists of holding a series of annual workshops for instructors that introduce them to software testing and the learning resources available through WReSTT. The main objective of the instructor workshops is to provide a forum where CS/IT instructors can improve their knowledge of software testing and software testing tools to support pedagogy. The outcome for this objective is that instructors participating in a workshop will increase their knowledge of software testing and the use of software testing tools to support pedagogy. More specifically, it is expected that there be an improvement of at least 30% between the pretest and posttest scores of the workshop participants.

Our first workshop was conducted in the spring of 2009 and was attended by 17 instructors from various colleges and universities. The title of the workshop was "The First Workshop on Integrating Software Testing into Program-

ming Courses (CS1-CS3) (WISTPC 2009)". The workshop lasted for two days and during that time the instructors were exposed to the basic concepts of testing and the resources available on WReSTT. The resources on WReSTT presented during the workshop included: EclEmma [17], JUnit [15], SWAT [32], Rational Functional Tester [18] and Web-CAT [9]. Web-CAT is a plug-in-based web application that supports electronic submission and automated grading of programming assignments. The instructors attending the workshop were also introduced to features of WReSTT to support pedagogy including, registration, techniques to browse the tools, and posting to the forums. We present the results of the first workshop in Section 4.

3.3 Integrating Testing into Programming Courses

Phase three of our approach consists of two components. The first component is to perform experiments on integrating testing into programming courses at several academic institutions and making an evaluation of the integration process. The second component involves developing learning materials that can be used by other institutions, and embarking on a broad dissemination plan to get other institutions involved. All learning materials developed will be made available on WReSTT.

The main objectives of integrating testing into programming course are: (a) improve the students' conceptual understanding of the approaches used to test software; and (b) improve their practical software testing skills with respect to the testing tools in WReSTT. The expected outcomes of this objective are: (a) at least 80% of the students exposed to integrating testing into programming courses will be able to describe at least two approaches used to test programs using automated tools; and (b) at least 80% of the students will be able to demonstrate how to use at least two of the automated tools that support the testing techniques in (a).

The research team at Florida International University conducted studies on integrating testing into the CS2 and SE courses with the support of the resources in WReSTT. The general approach used in both courses was non-intrusive, that is, the courses were taught similar to previous classes, but the students in the treatment group were exposed to the learning material available in WReSTT.

In the CS2 course a teaching assistant (TA) was assigned to the course and the TA's responsibilities included: (1) teaching students how to develop unit test cases for their assignments; (2) teaching them how to use the results from code coverage to improve testing their programs; and (3) demonstrating how to use the learning materials for the testing tools on WReSTT. The TA was not available to help the students with understanding their assignments, since the students were expected to use the normal resources for the course, e.g., Professor and other course TAs. In the SE course WReSTT was introduced to the students prior to the topic of software testing being covered. The only requirement added to the course, to encourage students to use the

Question	Instructors' Workshop		Software Engineering Course	
	Pretest	Posttest	Pretest	Posttest
2. Have you ever used tools to support testing of programs?	N(7) Y(5)	N(1) Y(11)	N(15) Y(3)	N(6) Y(12)
3.b.i Unit Testing Tool Proficiency	Avg = 2.7 (6 responses)	Avg = 3.5 (8 responses)	Avg = 4 (1 response)	Avg = 1.7 (8 responses)
3.b.ii Web-based Testing Tool Proficiency	Avg = 3.5 (2 responses)	Avg = 2.6 (7 responses)	NA	NA
3.b.iii Functional Testing Tool Proficiency	Avg = 3.7 (3 responses)	Avg = 2.7 (6 responses)	NA	NA
3.b.iv Code Coverage Tool Proficiency	Avg = 3 (3 responses)	Avg = 3.3 (6 responses)	(0 responses)	Avg = 3 (4 responses)
4. Do you know of any online resources that provide information on software testing?	N(7) Y(5)	N(2) Y(10)	N(11) Y(7)	N(5) Y(13)
6. How beneficial do you think it is to use tools to support the testing of programs?	Avg = 3.9 (11 responses)	Avg = 4.6 (11 responses)	Avg = 4.3 (18 responses)	Avg = 4.6 (18 responses)
8. How well do you know any automated grading tools that encourage students in CS1- CS3 to test their programs before submission?	Avg = 2.2 (12 responses)	Avg = 2.8 (12 response)	NA	NA

Table 1. Results for the closed ended questions in the pretest/posttest instrument. N − No; Y − Yes; Avg − average of scores are out of 5; NA − Not Applicable in this study.

resources in WReSTT, was awarding bonus points to those student teams that used tools to automate the testing of the code for their projects. These tools needed to support unit testing and code coverage. We report on these studies in next section.

4. Case Study

In this section we describe the evaluation studies performed during WISTPC 2009[2], the Fall 2009 CS2 class and the Spring 2010 SE class. The evaluation for WISTPC 2009 focused on the effectiveness of improving the participants' knowledge of software testing and the use of software testing tools to support pedagogy. The studies in the CS2 and SE classes focused on improving the students' practical skills of testing programs by using automated testing tools. In the following sections we describe the instruments used to capture the data, present a summary of the results obtained, and discuss issues related to the evaluation for the studies.

4.1 Data Capture

Workshop: The data collected for the study reported in this paper is based on a pretest/posttest instrument that was administered to the participants of WISTPC 2009. The instrument consisted of nine questions that use both closed and open ended questions. There were four classes of questions: Q(1)-(3) focused on program testing and the use of testing tools, Q(4)-(5) on online resources available to support testing, Q(6)-(7) assessed the importance of tool support for software testing, and Q(8)-(9) knowledge of automated grading tools that encourages testing. The leftmost column of Table 1 shows the closed ended questions from

the pretest/posttest instrument. The instrument was administered to the participants during the introduction session and closing session of the workshop. Seventeen instructors attended the workshop and they came from a cross-section of tertiary US educational institutions. Other evaluation instruments were administered during the workshop but we do not describe them in this paper since they focused on the logistics of the workshop and effectiveness of the presenters.

CS2 and SE Classes: The instruments used to collect data from the students in the CS2 and SE classes included pretest/posttest and for the SE class the grading rubric used during the demonstration of the class project. Thirty-one (31) students in the CS2 class volunteered for the study (treatment group). In the SE class eighteen (18) students participated in the study and were divided into eight (8) teams for the class project. There was no control group for the SE class. The pretest/posttest instrument consisted of seven questions, which were similar to the questions used in the instructors' workshop. The grading rubric contained criteria on the testing tool(s) used, and the type of tool(s) selected i.e., unit testing tool and/or code coverage tool. The testing tools were used mainly to support unit testing of classes during the implementation and testing phases of the project.

4.2 Results

Workshop: The effectiveness of the workshop was evaluated using a pretest/posttest instrument that included questions such as the participants' knowledge and proficiency of various tools and their perceptions of the usefulness of tools to support program testing. Table 1 shows the results for the closed ended questions used in the instrument. Columns 1 shows the question text, Columns 2 and 3 show the results for the pretest and posttest, respectively. Rows with ques-

Question	Instructors' Workshop			
	t	df	sig	Cohen's d
3.b.i Unit Testing Tool Proficiency	-2.449	4	0.070	-1.095
3.b.iii Functional Testing Tool Proficiency	0.200	1	0.874	0.141
3.b.iv Code Coverage Tool Proficiency	1.000	1	0.500	0.707
6. How beneficial do you think it is to use tools to support the testing of programs?	-3.730	10	0.004	-1.125
8. How well do you know any automated grading tools that encourage students in CS1-CS3 to test their programs before submission?	-2.390	10	0.038	-0.721

Table 2. Results after applying paired sample t-tests and Cohen's d effect size on the data collected.

tion numbers 2 and 4 show the number of responses with the value no (N) and yes (Y). The other rows in the table contain the average value (out of 5) for the responses given by the participants. For these questions the Likert scale of 1 to 5 was used, where 1 indicated barely competent, not at all beneficial, or no knowledge of the tool as the case may be, and 5 indicated extremely proficient, extremely beneficial, and very high knowledge about the tool.

At the end of the workshop, the percentage of participants who were exposed to tools that support testing of programs had increased from 58.33% to 91.67%. Similarly, the percentage of participants who were aware of online resources that provide information on software testing increased from 58.33% to 83.33%. Paired samples t-tests and Cohen's d effect size [4] were conducted and calculated respectively to test and measure the difference in the proficiency of using various tools, the perceptions of usefulness of the tools, and knowledge of automated grading tools. Owing to the small sample size and the use of several univariate statistical significance tests, the p-values of the tests were not used for inference [28]. Instead the p-values were simply used as indicators of a possible difference between the pretest and the posttest measurements and the Cohen's d values were used to quantify the magnitude of the difference.

As can be seen from the Table 2 Columns 2 through 5, the perceptions of participants about the usefulness of the tools to support program testing had the maximum effect size (1.12), indicating that there was a positive change of about 112% from before the workshop to after the workshop that the tools are indeed beneficial. Similarly, in their opinion, participants overall became more proficient in the use of unit testing tools (d=1.09) and gained more knowledge in automated grading tools that encourage their students to test their programs before submission (d=.72). However, participants felt that their proficiency level in using the functional testing tool and the code coverage tool worsened after attending the workshop. This is indeed an area of concern.

CS2 and SE Classes: Initially, 31 students volunteered for the study (treatment group) in the CS2 class. Of these, 10 students created user accounts on WReSTT, and the 10 accounts were accessed after registration. None of the students completed the study. It was apparent that students were not motivated to continue using the WReSTT testing resource in completing their assignments unless this was a course requirement and awarded credit for using the testing tools.

The results captured using the pretest and posttest for the SE class are shown in Table 1 Columns 4 and 5. The results show that after exposure to WReSTT there was a significant increase (400%) in the number of students that used tools to support the testing of programs. The contents of Table 1, row with question number 4, indicates that after the treatment more students (180% increase) were able to identify at least one online resource that provided information on software testing. The results for the SE course shown in the table were not statistically significant due either to the sample size or the effect of exposing students to the resources in WReSTT.

The data obtained from the grading rubric and observation of the project demonstrations indicated that five of the eight teams used testing tools during the validation of the team project. The five teams all used unit testing frameworks, these frameworks included JUnit [15], MbUnit [29], and Visual Studio Team System 2008 [26]. Two of the teams also used code coverage tools during testing, these tools included Cobertura [3] and the code coverage tool in Visual Studio Team System 2008 Test Edition [26]. The teams that used the unit testing tools did a better job at writing test cases and testing the overall system. In addition, these teams were more efficient when it came to demonstrating the execution of test cases i.e., they were able to execute the test drivers without any problems. The two teams that used the code coverage tools were able to explain the adequacy of the test cases with respect to the statement coverage achieved during testing. The students in these teams also stated that they were not able to get 100% code coverage due to the time constraints, that is, they did not have enough time to write the number of test cases required to get 100% code coverage.

4.3 Discussion

The expected outcome of the workshop was to improve the participants' knowledge of software testing and the use of software testing tools to support pedagogy by at least 30%. This outcome was achieved for the most part as stated in the previous subsection, specifically in the areas of (1) using tools to support program testing, and (2) being aware of online resources to support software testing. Although not shown in Tables 1 and 2 most participants were able to identify at least one tool to support unit testing, web-based testing, functional testing and code coverage upon completion of the workshop. In addition, most of the participants were also able to identify at least one online resource containing testing resources.

A review of the answers to several of the questions on the pretest/posttest instrument revealed that some of the question may have been ambiguous. In the pretest, participants identified unit testing tools, e.g., JUnit [15], as automated grading tools that encourage students in programming courses to test their programs before submission. The results of the posttest revealed that the presentation at the workshop of coverage tools may have to be reviewed. The participants identified JUnit as a coverage tool when performing program testing. This may have been due to the fact that EclEmma [17] was introduced directly after the JUnit presentation and participants did not see the distinction. The presentation on the functional testing tool, Rational Functional Tester [18], appeared to have been overwhelming for the participants.

Introducing students in the SE class to software testing tools addresses the second objective and expected outcome described in Section 3.3. Recall the objective is to improve the students' practical software testing skills with respect to the testing tools in WReSTT. The expected outcome is at least 80% of the students will be able to demonstrate how to use at least two of the automated tools that support the testing techniques. The results obtained during the demonstration of the student projects showed that we have not yet accomplished this objective since only 63% of the student teams (50% of the students) were proficient in using a unit testing tool. These numbers were even lower for the code coverage tool. The reason for this poor result may be attributed to the fact that the course was taught by a PhD student who was teaching the course for the first time.

We plan to repeat the study in a future SE class with a more experienced instructor and continue to offer students incentives for using tools to support testing. We are also modifying the interface to the WReSTT web site to incorporate some of the social networking concepts such as allowing students to create a profile and a list of friends that may participate in testing tutorials. We plan to repeat CS2 class study in Fall 2010 and will integrate testing topics explicitly into the syllabus. Both the instructor and the teaching assistant for the course in the fall attended the recently held workshop for integrating testing in to programming courses (WISTPC 2010).

5. Related Work

Although instructors have been able to introduce testing into their programming courses with some success, there has been limited focus on providing instructors with the knowledge, tools, and guidelines necessary to make such a transition smooth and minimally intrusive. In Section 2 we reviewed several of the works related to integrating testing into programming courses and will not repeat them here. To the best of our knowledge, this is the first work that aims to provide both a central repository of software testing tools, and the training workshops necessary to help instructors successfully integrate these tools into their CS1-

CS3 and SE courses. However, some researchers have proposed approaches similar to ours which we now describe as related work.

There have been several workshops and user group sessions on teaching software testing in the computer science course curricula [10, 22, 23]. The annual Workshop on Teaching Software Testing (WTST), Kaner et al. [22, 23], is concerned with teaching many of the practical aspects of university-caliber software testing to both academic and commercial students. Both academics who have experience teaching testing courses, and practitioners who teach professional testing seminars share their experiences and techniques with other instructors and graduate students. The Web-CAT User Group meetings, led by Edwards [10], allow users of an automated grading tool for programming and testing-related assignments to exchange their experiences using the tool, while attracting instructors that may be thinking of adopting the tool. Although not specifically geared towards introductory CS courses, such forums are similar to the WReSTT workshops as they also provide avenues for instructor education on effective software testing techniques and tools.

Lastly, many repositories and tutorials on software testing tools are accessible via the World-Wide Web [12, 25, 34]. These repositories contain a plethora of software testing tools and resources. However, the vast number of testing tools provided by these repositories make them inconvenient for use by instructors. WReSTT offers a practical solution by narrowing the scope of the tools and tutorials to those that both instructors and students have found useful in programming courses.

6. Concluding Remarks

In this paper we presented an approach to integrate testing tools into programming courses by providing a web-based repository of software testing tools, training instructors in the area of testing techniques and tools, and integrating the use of testing tools in various programming courses. The contents and structure of the web-based repository of software testing tools (WReSTT) was described in the paper. We have presented the results of the first instructors' workshop and the first study on integrating testing tools into CS2 and SE courses. We have recently held the second instructors' workshop and are currently performing additional studies in SE classes. Although the first set of results are promising, we are continuing to investigate innovative ways to integrate software testing into programming courses. We expected to redesign the WReSTT site to be more attractive to students and provide them with a features that support team-oriented learning.

Acknowledgments

References

[1] M. Clark. JDepend, May 2010. `http://www.clarkware.com/software/JDepend.htm`.

[2] "CNSS". Software 2015: A national software strategy to ensure u.s. security and competitiveness. Technical report, Center for National Software Studies, 2005.

[3] Cobertura Team. Cobertura, May 2010. `http://cobertura.sourceforge.net/`.

[4] J. Cohen. The earth is round (p < .05). *American Psychologist*, 49(12):997–1003, December 1994. `http://web.math.umt.edu/wilson/Math444/Handouts/Cohen94_earth%20is%20round.pdf`.

[5] C. Desai, D. S. Janzen, and J. Clements. Implications of integrating test-driven development into cs1/cs2 curricula. *SIGCSE Bull.*, 41(1):148–152, 2009. ISSN 0097-8418.

[6] Drupal Community. Drupal, 2008. `http://drupal.org/`.

[7] S. H. Edwards. 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)*, pages 148–155, New York, USA, 2003. ACM Press.

[8] S. H. Edwards. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE Conference*, pages 26–30, New York, NY, USA, 2004. ACM. ISBN 1-58113-798-2.

[9] S. H. Edwards. Web-CAT: the Web-based Center for Automated Testing, 2009. `http://web-cat.cs.vt.edu/`.

[10] S. H. Edwards and M. A. Perez-Quinones. Web-cat user group, March 2008. BOF session at the 39th SIGCSE Technical Symposium on Computer Science Education.

[11] S. Elbaum, S. Person, J. Dokulil, and M. Jorde. Bug hunt: Making early software testing lessons engaging and affordable. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 688–697, Washington, DC, USA, 2007. IEEE Computer Society.

[12] D. Faught. TestingFAQs.org - an information resource for software testers, 2010. `http://www.testingfaqs.org/`.

[13] M. Feathers. CppUnit, May 2010. `http://apps.sourceforge.net/mediawiki/cppunit/`.

[14] S. Frezza. Integrating testing and design methods for undergraduates: teaching software testing in the context of software design. *Frontiers in Education, Annual*, 2:S1G1–4, 2002.

[15] E. Gamma and K. Beck. JUnit, 2008. `http://www.junit.org/`.

[16] M. H. Goldwasser. A gimmick to integrate software testing throughout the curriculum. In *Proceedings of the 33rd SIGCSE Conference*, pages 271–275. ACM, 2002.

[17] M. R. Hoffmann. EclEmma, 2008. `http://www.eclemma.org/`.

[18] IBM. Rational Functional Tester , 2008. `http://www-01.ibm.com/software/awdtools/tester/functional/`.

[19] U. Jackson, B. Z. Manaris, and R. A. McCauley. Strategies for effective integration of software engineering concepts and techniques into the undergraduate computer science curriculum. In *SIGCSE '97: Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*, pages 360–364, New York, NY, USA, 1997. ACM.

[20] D. S. Janzen and H. Saiedian. Test-driven learning: intrinsic integration of testing into the CS/SE curriculum. *SIGCSE Bull.*, 38(1):254–258, 2006.

[21] E. L. Jones. Integrating testing into the curriculum — arsenic in small doses. *SIGCSE Bull.*, 33(1):337–341, 2001.

[22] C. Kaner, S. Barber, and R. Fiedler. Workshop on teaching software testing: Wtst 7, Jan. 2008. `http://www.wtst.org/wtst7.html`.

[23] C. Kaner, S. Barber, and R. Fiedler. Workshop on teaching software testing: Wtst 8, Jan. 2009. `http://www.wtst.org/wtst8.html`.

[24] T. C. Lethbridge, J. Diaz-Herrera, R. J. J. LeBlanc, and J. B. Thompson. Improving software practice through education: Challenges and future trends. In *FOSE '07: 2007 Future of Software Engineering*, pages 12–28, Washington, DC, USA, 2007. IEEE Computer Society.

[25] M. J. Lutz, W. M. McCracken, and S. Mengel. Swenet - network community for software engineering education, Sept 2009. `http://www.swenet.org/`.

[26] Microsoft Corporation. Visual Studio Team System 2008, May 2010. `http://msdn.microsoft.com/en-us/library/ee338734(v=VS.90).aspx`.

[27] G. J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, second edition, 2004. ISBN 0471469122.

[28] P. Natesan and B. Thompson. Extending improvement-over-chance i-index effect size simulation studies to cover some small-sample cases. *Educational and Psychological Measurement*, 67(1):59–72, 2007.

[29] NUnit.org. MbUnit, May 2010. `http://www.mbunit.com/`.

[30] RTI. The economic impacts of inadequate infrastructure for software testing. Technical Report 7007.011, National Institute of Standards and Technology NIST, May 2002.

[31] S. Schaub. Teaching cs1 with web applications and test-driven development. *SIGCSE Bull.*, 41(2):113–117, 2009.

[32] Ultimate Software. SWAT, 2009. `http://sourceforge.net/projects/ulti-swat/`.

[33] Wikipedia. xUnit, 2009. `http://en.wikipedia.org/wiki/XUnit`.

[34] L. Williams and S. Heckman. OpenSeminar - software testing resources, 2010. `http://openseminar.org/se/modules/7/index/screen.do`.

[35] WReSTT Team. WReSTT: Web-based Repository for Software Testing Tools, 2009. `http://wrestt.cis.fiu.edu/`.