Using a Real World Project in a Software Testing Course

Daniel E. Krutz, Samuel A. Malachowsky, and Thomas Reichlmayr
Software Engineering Department
Rochester Institute of Technology
1 Lomb Memorial Drive
Rochester, NY 14623
{dxkvse, samvse, tjrese}@rit.edu

ABSTRACT

Although testing often accounts for 50% of the budget of a typical software project, the subject of software testing is often overlooked in computing curriculum. Students often view testing as a boring and unnecessary task, and education is usually focused on building software, not ensuring its quality. Previous works have focused on either making the subject of testing more exciting for students or on a more potent lecture-based learning process.

At the Department of Software Engineering at the Rochester Institute of Technology, recent efforts have been focused on the project component of our Software Testing course as an area of innovation. Rather than previous methods such as a tightly controlled and repetitive testbed, our students are allowed to choose a real-world, open source project to test throughout the term. With the instructor as both counsel and client, students are expected to deliver a test plan, a final report, and several class-wide presentations.

This project has achieved significant student praise; qualitative and quantitative feedback demonstrates both increased satisfaction and fulfilled curricular requirements. Students enjoy the real-world aspect of the project and the ability to work with relevant applications and technologies. This paper outlines the project details and educational goals.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computers and Information Science Education- Computer science education; Curriculum

General Terms

Design, Reliability, Verification

Keywords

Software Testing, Software Engineering Education, Software Project

SIGCSE'14, March 3–8, 2014, Atlanta, GA, USA.

1. INTRODUCTION

Software testing is an important aspect of creating reliable software. Defects can have numerous adverse affects on an application, ranging from unhappy customers and users to injury or even death. Projects should place a more significant focus on testing; recent studies have indicated that testing makes up over 50% of the cost of a typical software project [15] and a third of the cost of all software bugs could be eliminated through improved testing [21]. Unfortunately, testing is far too often overlooked in both industry and academia.

Software Engineering education is far too often deficient in this area as well. Although most universities have programs in computing, very few offer courses with a primary focus in software testing [11] [14] [18]. Additionally, the educational mindset of most schools is to teach students how to build software, not break it [13]. Largely due to the difficulty of grading such a component or a lack of course time, testing often comprises only a minor portion of the student's grade in the majority of programming courses [19] [10]. Students often lack enthusiasm about testing; they are usually much more interested in building software than in ensuring its quality [1]. Finally, testing as both a concept and a practice can be very difficult to teach through lectures. Kaner *et al.* [9] stated that "the challenge is to develop group activities that can foster insight—a level of abstract understanding that can apply from situation to situation—rather than emphasizing detailed procedural understanding."

There is a significant amount of research which shows the importance of software testing, with a substantial amount of improvement needing to be done in both academia and industry [15] [12]. Additionally, software testing in our educational institutions needs to have a more practical focus with educators doing a better job readying their students for industry [20] [8].

At the Rochester Institute of Technology (RIT), we have offered an upper division Software Testing course since 1998, with a wide focus including process, tools, and analysis. A project component has always been a significant portion of this course, but it became obvious that improvements needed to be made. We have recently altered this project to more directly address a lack of student enthusiasm regarding testing, the desire for students to work on a real-world project, and the need for students to become better acclimated with contemporary testing technologies.

In this paper, we describe an innovative approach to teaching software testing through a project component. Under the supervision of the instructor, who also plays the role

as customer, small student teams first choose a moderately sized open source project. Each team then develops a test plan to be followed for the rest of the project term and proceeds to test the chosen application using this plan. Teams give presentations throughout the term to report their progress. The project culminates with a final presentation and test report.

This updated project component has been included in several class offerings and has had substantial success. Students have stated that it has not only significantly increased their knowledge of software testing as a discipline, but of its value as well. Feedback has shown praise both for the real-world nature of the project and the reinforcement it gave to the need for testing.

The remainder of this paper is organized as follows. Section 2 describes the Software Testing course with its general structure and main components. Section 3 provides an overview of the project. Section 4 discusses a sample student project. Section 5 discusses project results. Section 6 offers student feedback based on the project. Section 7 describes related work. Section 8 discusses future work on the project, and section 9 provides a summary of this work.

2. ABOUT THE COURSE

Primarily comprised of upper division Software Engineering students, the Software Testing course is also offered to other programs including Computer Science, Computer Engineering, Electrical Engineering, and Game Design. The only prerequisite for our testing course is an earlier class which introduces basic concepts of software engineering including teamwork, development processes, and proper documentation. In this prerequisite Introduction to Software Engineering course, students have already been introduced to some aspects of testing, including unit testing, system testing, integration testing, and acceptance testing. In many cases, other non-required courses and a required one-year cooperative internship (co-op) have exposed students to various aspects of testing as well. The Software Engineering department considers the 3-credit Software Testing course to be the primary means of fully exploring software testing and closely related disciplines.

The course has three primary learning outcomes: to instruct students on the importance of software testing, to expose them to various testing-related tools and techniques, and to explain and exercise the decision process and situations in which certain tests should and should not be used. While various tools such as jUnit[1], DJUnit[2], jMock[3], and Selenium[4] are discussed throughout the course, it is their underlying categorization and appropriate use that is emphasized. The studies regarding the importance of testing are expounded both by looking at use cases that illustrate a need for testing and by exploring why testing is often neglected by development teams.

While most students do not become overly enthralled with software testing, we do expend effort to dispel the myth that testing is the bland, monotonous field that many believe it to be. While approximately half of each classroom session is devoted to a lecture, the remaining time is reserved for a relevant hands-on activity or discussion. As an example, each class begins with an interactive "Bug of the Day" (BotD) activity [11] in which students study a real world software bug, its impacts, and the ways it could have been avoided. This activity is designed to highlight common bugs and practical application of course topics to actual examples.

Students are graded on several criteria. Each term there are three exams, several homework assignments, and a project based component. Class size is typically 25-40 students.

3. ABOUT THE PROJECT

The Software Testing course at RIT has always had a significant project component. Previously, the project involved all student groups being assigned the same set of code in which they were expected to find known bugs. The tools and testing techniques they were allowed to use was restricted, and the project was largely viewed as bland and unexciting for the students. They did not enjoy working on the same cookie cutter project, wanted something more realistic, were interested in making a real-world contribution, and wanted something that would look better to potential employers. In order to address these concerns and to meet educational goals, we decided to create a new project for the course.

During the first week of the 15 week term, teams are formed. Students are able to assist in the team creation process by choosing which other students they would like to be partnered with. Team size is targeted to 4-6 students, as this is often the size of groups in industry and has been found to be conducive to student learning in previous research [6] [16].

Once the teams have been formed, their first step is to choose an application to be tested. Students are asked to select medium-sized open source applications for testing. They must be large enough to provide an adequately sized test bed, but not so large as to overwhelm the teams. Applications with roughly 100-200 classes have been found to be appropriate for this project, and students are not limited by programming language, even if the instructor is not familiar with the chosen language. Projects with a high level of documentation are preferred, as this provides a basis for acceptance tests and the documentation itself can also be tested. The final step in the application selection process is to have the program approved by the instructor. This must be done by week 3.

The next major deliverable, due in week 5, is the project test plan. Students are provided with an initial template, but test plans are expected to significantly differ from one another due to the wide variety of applications selected. Groups are encouraged to make the best test plans possible, but are not necessarily held to them during the remainder of the course term. Because of expected but unforeseen hurdles as well as new concepts covered in class, students are allowed to build on to and deviate from this test plan as necessary.

In week 6, each team presents their test plan, along with any results already gathered by the team. This gives the teams an opportunity to receive feedback from their peers, gain valuable public speaking experience, and exposure for their classmates to new testing tools and techniques (which they may use on their own projects). After the presentation, a question and answer session takes place in which students must often defend the choices made for their project. Addi-

[1]http://junit.org/
[2]http://works.dgic.co.jp/djunit/
[3]http://jmock.org/
[4]http://docs.seleniumhq.org/

Table 1: Discovered Defect Types

Week	Deliverable
1	Teams Formed
3	Application Chosen
5	Test Plan
5-15	Plan Execution
6	Plan Presentations
10	Progress Report
15	Final Report & Presentation
15	Postmortem

tionally, students must write a 2-4 page reflection document describing what has and has not gone well so far. A primary objective of this reflection is to allow the students to understand and continue to build upon their strengths while also identifying and remedying areas of improvement. Groups are expected to perform a critical analysis of these findings and propose resolutions for problematic areas.

In week 10 of the term, groups are asked to submit a brief report detailing their progress to that point. This deliverable serves two purposes: it gives the instructor insight with an opportunity to intervene, and it provides a milestone for groups mid-project.

In the final week, students submit their final report, which details not only the findings of their testing, but also the specific tests used and relevant data. Students are also encouraged to state why they chose and carried out particular tests and provide a rationale for the processes and techniques they employed. They are not expected to propose resolutions for the discovered bugs, as this could violate the intent of the exercise and serve to distract the team. Groups are, however, encouraged to identify the steps needed to duplicate the defect, which appropriately represents the similar action needed in industry to resolve defects. Much like in week 6, students again present their findings to the rest of the class, including a 5-10 minute window for questions and answers.

The final project deliverable is a postmortem. In this document, students reflect on what went well, what went poorly, and how they could improve upon each. This critical analysis is also expected to contain possible resolutions for all difficulties encountered along the way.

Grading of all deliverables is done using a rubric. The presentations, for example, are evaluated on how well they prepared, the quality of project information conveyed to the class, and how well they answered project questions. No significant part of the grading is based upon the number of bugs found by a team, as students are encouraged to be more concerned with the processes and techniques they are using rather than the number of defects they discover.

4. SAMPLE PROJECT

As stated in the last section, the primary criteria for selection were size and a restriction to open-source projects. Students selected a wide range of open source applications, including both traditional and mobile applications, applications written for Unix, Windows, and OSX operating systems, and applications written by both small groups and organizations the size of Mozilla. A few of the tested applications included JBrick [5], tuxGuitar [6], Notepad++ [7] and components of Firefox [8].

In the following example, we will discuss an example student testing project which analyzed jBrick, an application designed to assist handicapped students in programming Lego robots. The group was comprised of 6 students, 3 of which had previously indicated that they wished to work together. Because one of the team members had previous experience with jBrick, it was chosen as the test application after a brief team discussion.

The students were able to gather both the source code and a significant amount of documentation from the jBrick project website. Their initial plan was largely devoted to usability and acceptance testing of the project. While these tests would have yield some results, instructor feedback indicated that they should explore a wider variety of applicable tests, keeping in mind the cost/benefit trade-off in their choices.

The team ultimately decided to add unit and mutation testing. They met twice a week for 1-2 hours and stayed in touch via text, Google Docs, and social media. The team regularly met with the instructor who played both the role of teacher and customer for the project. During their final presentation, they indicated that they had performed most of their intended testing on the application, but were unable to perform some of the unit tests due to time constraints and some of the mutation testing due to technical limitations.

The team's testing revealed some compelling results. Since jBrick was intended for both the visually and non-visually impaired, usability testing was performed on the application using representatives of both groups. While some usability issues were identified by visually impaired users, a surprisingly higher number were found by non-impaired users. Additional issues found included negative application performance due to poorly written code and functions which needed repair (revealed in unit testing).

In their postmortem, the team indicated that although they believed their project went well, they felt there was room for significant improvement in both their testing methodologies and their team dynamics. The team expressed that they should have used more automated testing tools to reduce the monotony of some of the tests as well as a wider variety of tests, including fuzz testing. The team also stated that they should have met more as a group and stayed in closer contact with one another throughout the course of the project. Members indicated that lack of communication significantly hindered their team's performance.

5. PROJECT RESULTS

Students have uncovered a wide variety of bugs with varying levels of severity. Table 2 shows some of the types of bugs discovered by 18 groups spanning several course offerings with each group testing a unique project. Instructors should remember that all projects are different and that the types and number of defects students discover will significantly vary. All examined projects are different and a properly developed testing process does not necessarily lead to the discovery of a large number or variety of defects.

[5]http://code.google.com/p/jbrick/
[6]http://sourceforge.net/projects/tuxguitar/
[7]http://notepad-plus-plus.org
[8]http://www.mozilla.org/en-US/firefox/

Table 2: Discovered Defect Types

Type of bug	# Found
Minor, non-functional	153
Incorrect Documentation	287
Poor Usability	113
Incorrect Functionality	49
Crash	35

The majority of issues discovered have been relatively minor, such as grammar mistakes, inadequate documentation, and poor usability. In most cases, however, students were able to find significant numbers of more serious issues such as crashes in the application and functionality that was incorrect. In measuring the severity of defects, teams studied the accuracy of results and if requirements were met or failed.

Students were able to find bugs using a wide variety of testing tools and technologies. Table 3 includes a list of those most frequently chosen by the students. The majority of teams chose to conduct unit, acceptance, usability, and compatibility testing. While these tests were certainly good choices, it is likely that teams selected them because they were both relatively easy to implement and students had been exposed to them early in the course.

Table 3: Types of Testing Conducted by Teams

Type of test	# Found
Unit	16
Acceptance	17
Usability	17
Fuzz	10
Accessibility	8
Security	3
Performance	10
Compatibility	18

After completion, most teams chose to submit their bug reports to the application's development team. Several have received messages from the developers thanking them and confirming the existence of the errors they had discovered.

6. STUDENT FEEDBACK

Students have expressed a significant amount of satisfaction in this project and it has contributed to their overall satisfaction with the course. At the conclusion of the term, students are asked to submit an anonymous survey rating several aspects of the course, instructor, and project. Several of these questions and student responses are shown in Table 4. These questions have been posed to students in the last three course offerings, all of which have used this project component. A total of 78 students from these sections have responded to the survey.

These results indicate that the vast majority of students not only enjoyed the project, but would also recommend it to a classmate as well. Additionally, most students felt that it reassembled a project which they were likely to encounter in the real world and were similar to tasks they were asked to complete while on co-op. Finally, students were asked

Table 4: Student Responses

	Yes	No	Total
Did you enjoy the software project?	66	12	85%
Would you recommend this project?	60	18	77%
Resembles a real world project?	60	18	77%
How much did you learn? (0-5)	-	-	4.15

to rate how much they learned from the project on a 0-5 scale, with 0 representing them learning very little and 5 representing the maximum amount of learning. The average student response was a 4.15/5 indicating that, on average, students felt the project to be very educational.

The end of term feedback also allowed students to write some of their thoughts regarding the course project. The following are samples of written feedback that have been received:

> "The Software Testing course helped to drive home the importance of testing at every stage in the software development life cycle. The project component was especially helpful in this area. We were able to see the abundance of errors that existed even in our favorite applications."

> "Testing one of my favorite open source applications helped to keep me interested in the project. Not only was I able to examine the source code of a tool I regularly used, but it made me feel good to know that I was assisting in removing defects from a tool I used on a regular basis, for both myself and other open source users as well. "

> "Testing is a difficult, time consuming and sometimes monotonous process. It is however, very important. I was shocked by all the bugs our team discovered in what I thought to be a heavily tested and defect free application."

> "Like with many aspects of software engineering, software testing is heavily process based. Before this project, I viewed testing as an ad-hoc process that waited until the conclusion of a project. This project has taught me the importance of documentation and process in software testing. Additionally, I've learned that testing should occur at numerous stages in software development."

After taking the course and returning from co-op, students have stated they this course project prepared them very well for projects which they worked on in industry. The project has also helped to increase specific interest in software testing for several students, some of which have subsequently taken full time jobs in the field of software testing. Many students have reported that they were able to gain a co-op or full time job in a field of software testing or a related discipline largely from their experiences with this project.

The enthusiasm students have about the project has also been stated as a major reason why many decide to enroll in the class. One testament to this is the growing enrollment figures for the elective course. The first term in which

the new project was offered was in the Fall of 2010. Since this offering, student enrollment figures have progressively increased for the course. The second offering of 2012 saw the maximum number of students allowed for a course, 40, to be reached, leaving numerous students on the waiting list unable to attend. While it is impossible to unequivocally state that the project is the sole reason for the increased student enrollment figures, we believe that these numbers act as yet another testament for student enthusiasm regarding the project. It is possible that the enthusiasm of the class instructors regarding the new project and its inherent variative nature has been a factor in student satisfaction as well. A chart representing student enrollment for the course since 2008 is shown in Figure 1. The line in the chart represents when the project was first offered as part of the course.

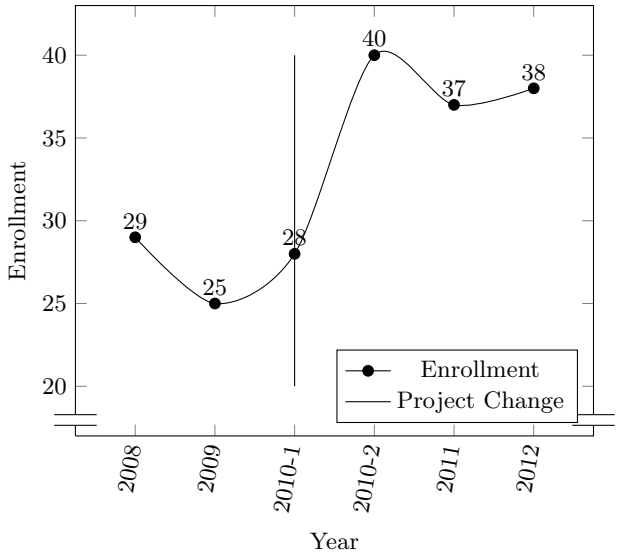

Figure 1: Student Enrollment Change

7. RELATED WORK

There have been numerous works which have stated the need of further emphasis on software testing education [1] [9] [2]. Some the deficiencies in software testing education include the inability to excite students about the topic, lack of relevant projects for students to interact with, and a primary educational focus on building rather than breaking software [13] [7] [1].

In order to assist software testing education, previous works have focused on enhancing the education process either through making it more exciting for students, or through a more educational experience. Gotel *et al.* [5] described the use of an open source web-based system for teaching software testing. This project revolved around having student-created problems which other students were expected to solve. Elbaum *et al.* [3] created a web-based tutorial known as Bug Hunt. The goal of this hands on activity was to teach students software testing at their own pace. Tutorials provided instant feedback to the user and automatically assessed the student's performance.

Preston [17] described the use of real-world projects in Information Technology education. The use of such projects was found to allow students to apply the theoretical knowledge they previously learned and solidify their understanding of the subject matter. Harrison [7] described a method of teaching software testing using viewpoints from both the developer and the software tester. This approach is beneficial since this allows students to see the importance and impact of proper software testing from each perspective. Goldwasser [4] described several fun and innovative methods of including software testing into a curriculum. In one example, students acted in a competitive fashion against a common test set.

8. FUTURE WORK

This project has been utilized in several sections of our Software Testing course and has been very successful, but there are enhancements which may be done to the project and further data which may be collected. One of the main purposes of the project is to allow a significant amount of freedom for each project team while achieving stated educational goals. While project deliverables are defined, each team is allowed to assign their own team roles, select their own projects, and choose the tests to run on the target application. The opportunity to make their own decisions and learn how to self-manage team in a supervised setting has presented some issues. Some of the teams have struggled with the extent of afforded freedom and have requested an extra deliverable and guidance during the term. In future implementations of this project, an extra deliverable will be added in week 10 which will include a written status and brief presentation. Finally, in past iterations some teams have been comprised of as many as 6 students; in the future, groups will be limited to 4-5 students to make meeting scheduling and communication demands more manageable.

This type of real world project is applicable to other educational areas of computing and may be applied to these courses as well. As an example, a software security course could implement a similar project to check for vulnerabilities in open source applications and propose fixes. A software design course could review the design of an application including the positive or negative ramifications of that design. In order to further measure the effectiveness of the project outlined above, we would like to poll seniors who took the Software Testing course regarding the effect of the project had on their subsequent experiences with software testing (i.e. co-op and senior capstone).

9. SUMMARY

Software Engineers need to understand the importance of developing high quality, defect free software. In order to ensure students are prepared for this task, we have developed an innovative software testing project which closely resembles tasks students will face upon graduation.

We have witnessed a significant increase in student enthusiasm in software testing as a subject and a discipline. This is largely due to us allowing students to select contemporary real world projects to test. Students have generally found the project to be very educational and feedback has indicated that it been a significant asset to them in industry. At least two other universities are planning to incorporate a similar project in their Software Testing classes and we encourage others to consider this approach in their courses as well.

10. REFERENCES

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

[2] S. H. Edwards. Teaching software testing: automatic grading meets test-first coding. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '03, pages 318–319, New York, NY, USA, 2003. ACM.

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

[4] M. H. Goldwasser. A gimmick to integrate software testing throughout the curriculum. *SIGCSE Bull.*, 34(1):271–275, Feb. 2002.

[5] O. Gotel, C. Scharff, and A. Wildenberg. Teaching software quality assurance by encouraging student contributions to an open source web-based system for the assessment of programming assignments. *SIGCSE Bull.*, 40(3):214–218, June 2008.

[6] J. Guo. Group projects in software engineering education. *J. Comput. Sci. Coll.*, 24(4):196–202, Apr. 2009.

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

[8] E. L. Jones and C. L. Chatmon. A perspective on teaching software testing. In *Proceedings of the seventh annual consortium for computing in small colleges central plains conference on The journal of computing in small colleges*, pages 92–100, USA, 2001. Consortium for Computing Sciences in Colleges.

[9] C. Kaner and S. Padmanabhan. Practice and transfer of learning in the teaching of software testing. In *Software Engineering Education Training, 2007. CSEET '07. 20th Conference on*, pages 157–166, 2007.

[10] F. Kazemian and T. Howles. A software testing course for computer science majors. *SIGCSE Bull.*, 37(4):50–53, Dec. 2005.

[11] D. Krutz and M. Lutz. Bug of the day: Reinforcing the importance of testing. In *Frontiers in Education*, 2013.

[12] C. Mao. Towards a question-driven teaching method for software testing course. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 5, pages 645–648, 2008.

[13] A. Meneely and S. Lucidi. Vulnerability of the day: concrete demonstrations for software engineering undergraduates. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1154–1157, Piscataway, NJ, USA, 2013. IEEE Press.

[14] G. J. Myers and C. Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004.

[15] L. Osterweil. Strategic directions in software quality. *ACM Comput. Surv.*, 28(4):738–750, Dec. 1996.

[16] D. Petkovic, G. Thompson, and R. Todtenhoefer. Teaching practical software engineering and global software engineering: evaluation and comparison. *SIGCSE Bull.*, 38(3):294–298, June 2006.

[17] J. A. Preston. Utilizing authentic, real-world projects in information technology education. *SIGITE Newsl.*, 2(1):4:1–4:10, Apr. 2005.

[18] T. Shepard, M. Lamb, and D. Kelly. More testing should be taught. *Commun. ACM*, 44(6):103–108, June 2001.

[19] J. Smith, J. Tessler, E. Kramer, and C. Lin. Using peer review to teach software testing. In *Proceedings of the ninth annual international conference on International computing education research*, ICER '12, pages 93–98, New York, NY, USA, 2012. ACM.

[20] J. A. Whittaker. What is software testing? and why is it so hard? *IEEE Softw.*, 17(1):70–79, Jan. 2000.

[21] W. Wong, A. Bertolino, V. Debroy, A. Mathur, J. Offutt, and M. Vouk. Teaching software testing: Experiences, lessons learned and the path forward. In *Software Engineering Education and Training (CSEE T), 2011 24th IEEE-CS Conference on*, pages 530–534, 2011.