A Software Testing Course for Computer Science Majors

Fereydoun Kazemian
Department of Computer Science
Rochester Institute of Technology
Rochester, New York 14623
fxk@cs.rit.edu

Trudy Howles
Department of Computer Science
Rochester Institute of Technology
Rochester, New York 14623
tmh@cs.rit.edu

Abstract

The ability to program is a fundamental skill for Computer Science (CS) majors, and most CS programs introduce programming concepts through a sequence of courses. In fact, "programming courses offer skills and training that meets many of the needs expressed by students, their near-term employers, and non-CS faculty" [3, p. 24]. However, in most of these courses, relatively little time is spent in teaching students how to program well and how to test and locate defects; much of the focus is in teaching language constructs, syntax, and basics of programming. As more topics are introduced and the discipline continues to broaden, it has become nearly impossible to add new topics or required courses without removing others. This paper documents the rationale and procedures in developing an undergraduate testing and debugging elective course for Computer Science majors. It identifies the methodology used to select available tools, and documents the rationale in designing the course and developing its outcomes.

Keywords: Software testing, validation and verification, debugging, defect prevention

1. Introduction

Over the last forty years, numerous system failures have been reported with catastrophic consequences in terms of cost and loss of life [16]. All these failures were traced to software defects. A 2002 study commissioned by the National Institute of Standards and Technology (NIST) found that software errors cost the United States economy about $59.5 billion annually [13]. One reason for this may be attributed to the lack of emphasis given to software testing during development. In 1996, the Workshop on Strategic Directions in Software Quality pointed out that software quality will become the dominant success criteria for the software industry [14]. Software testing is an important process in improving software quality.

According to the Computing Curricular 2005 (CC2005) document, one key role for computer scientists is to undertake challenging programming jobs and supervise other programmers to keep them aware of new approaches [4]. Computer scientists design and implement everything from system infrastructure software to applications and the ability to program is identified as an essential skill that must be mastered by anyone studying computer science [3].

In programming courses, students learn the syntax and constructs of a language along with algorithmic and design aspects of software development. Due to a lack of time, other related skills such as testing and dealing with quality issues are not given enough emphasis to ensure that our future computer scientists will have the ability to write "good" software. We define "good" software as software that not only works, but that also considers the many "ilities" including maintainability, reliability, testability, understandability, and reusability. Educators must face the challenge to equip students with the necessary skills and attitudes to effectively deal with software quality concerns [7, 8].

This paper documents the rationale and procedures in developing an undergraduate testing and debugging elective course for Computer Science majors. It identifies the methodology used to select available tools, and documents the rationale in designing the course and determining its outcomes.

2. Current Practices

In the CC2005 document, both the IEEE and the ACM emphasize the fundamental need for testing and debugging skills [4]. However, verification and validation, maintenance and quality issues are weighted relatively low in emphasis and are allocated a relatively small number of hours in the curriculum.

Allocating more curriculum hours is understandably difficult to do. The computer science discipline is always expanding. New topics are added and few seem to become obsolete; it becomes difficult to find time to cover what is already identified as critical topics. The CC2001 document

identified a dozen areas where recent technical advances had increased the importance of the topic.

In spite of all the new topic areas, programming remains a paramount skill. Matthew Szulik, CEO of Red Hat reports having trouble finding highly skilled programmers [10]. Thomas [17] reports that while the job market for computer scientists is still hot, employers are continuing to hire based on skills and not only on the basis of a degree.

Our department's current model for teaching testing is consistent with what is described by Marrero and Settle [11] – the importance of testing is often overlooked or overshadowed in the already full introductory programming courses. Although some courses require the development of a test plan, the plans are often incomplete, incorrect, or missing. Another observation is that when an emphasis on "thinking about testing before coding" requires the submission of a test plan, it rarely maps to the extent or content of testing that actually occurs; students are completing the test plan because it's required, but they don't see the usefulness of the plan when they reach the testing stage. Teachers of upper division courses note that many students lack a systematic approach to detecting defects and locating and correcting the problem, and few use the available tools to aid in debugging and testing.

We conducted an informal survey through the SIGCSE mailing list that revealed most CS departments infuse topics including some aspects of testing debugging, defect prevention, software quality, maintainability or code metrics into the introductory programming sequence; one department included an industry partner to provide a real-world flavor. A few programs allowed students to take courses offered through other departments (such as Information Systems or Software Engineering) as electives, and one reported teaching a quality assurance and testing course offered through the CS department.

Reviewed papers on software testing courses document course strategies and frameworks [1, 2, 6, 7, 8, 9, 11, 15]. Models vary in scope from approaches focusing on development only (coding), development and design, and full life-cycle phases. Kaner provides extensive documents, including assessment techniques at testingeducation.org.

3. Our Approach to Developing the Course

We set out to develop a unique approach in developing this course. Other frameworks seem to allow the infrastructure or tools to determine the course; instead, we decided that the course should determine the tools.

Although lecture sessions will follow the traditional approach to teaching testing and debugging, the content of several assignments and the scope of the capstone project will be determined by the student, with approval of the instructor.

Students possessing the prerequisites for this course (3 quarters of Java and 1 quarter of C++) are often able to identify their own weaknesses when it comes to testing or debugging skills. Some students are not confident in their ability to determine test cases and others are not skilled at determining a fix once a defect is discovered. Other students indicate they feel comfortable with testing and locating defects, but the structure and complexity of their code makes corrections difficult. In these cases, students will identify a specific skill and will work with the instructor to identify a useful project in which the skill will be learned and practiced.

We feel this approach provides useful knowledge for students that will be needed in upper division courses where they will be required to write more sophisticated and complex programs. These are also critical skills students will need to develop in preparation for future employment. The instructor will guide the student through the process in a systematic way so that debugging and testing are no longer ad hoc activities. It is also felt that skill building will improve interest in writing programs since students should be less anxious about the correctness of their implementation.

Because of the range and scope of problem areas we expect students to identify, it was decided that the key to implementing this course was to not limit ourselves to a single tool. The pursuit of commercially available tools was abandoned after months of working with marketing representatives and vendors. It became clear that it was not economically feasible to acquire the range of tools needed, and the licensing restrictions were limiting the course activities. As a result, we examined the available open source software (OSS).

We found a wide range of OSS tools including debuggers, profilers, complexity analyzers and testing software. Some are stand-alone applications and others are plug-ins to other popular tools and environments. The added benefit of OSS is that the source code is available; if a tool doesn't quite fit with the current problem, we have the option of initiating a student project to modify the code. Open source project developers will be informed of discovered defects and our solution to correcting them. This will give the students a sense of making a contribution to an OSS project while practicing testing techniques using an appropriate testing tool.

When students identify a capstone project topic, they will be required to review the available software and evaluate the functionality of possible candidates. The instructor will guide the student through the selection and evaluation process, a skill which in itself is a valuable learning experience.

In most introductory programming courses, a comprehensive project is the final deliverable. Because the assignment is evaluated at the end of the term, students seldom have an interest in determining their mistakes and are never asked to correct and resubmit a working version. We found this fosters a naïve attitude regarding correctness and accountability, one that would not be acceptable in a

real-world work environment. In preparation for the first offering of this course, student project submissions from previous programming courses have been archived, and these submissions will become the basis for an assignment in the debugging and testing course. Students will be asked to test, debug and evaluate both their own and other students' previous submissions.

When projects are developed, the instructor and student will also determine appropriate assessment criteria. Kaner [9] provides many assessment ideas along with grading notes.

4. The Selection of Tools

In designing this course an important consideration was the selection of appropriate testing and debugging tools. After reviewing many commercially available tools, we decided against using any of them because of the cost and licensing issues and the fact that the vendors were not willing to provide us with educational versions of these tools or as gifts-in-kind. We decided to look at the open source option and found that many such tools were available as either stand-alone or plug-ins to other Integrated Development Environments (IDE). To identify the testing tools we used SourceForge.net as our main repository of open source projects. Our overall selection criteria were simple. First, we identified the tools based on functionality, reliability and usability that would be appropriate for the course. We made a short list of the tools and read the available reviews for them. Using this list, we checked Freshmeat.net web site that provides some useful information on a large number of OSS projects. This information includes the date of last update, rating, vitality, popularity, type of license and version number. The definitions on rating, vitality, and popularity are given at Freshmeat.net.

5. Course Design

The course proposal defines a 4 credit hour, one quarter course. In the quarter system, this equates to 40 hours of instruction. The lecture components consist of the following topics:

- An overview of testing including a definition of defects, faults and errors, verification and validation, and static and dynamic testing. Administrative, budgetary, and political challenges along with specific issues for OO testing will also be covered.
- Static testing methods including desk checking, inspections and code reviews.
- Dynamic testing strategies including path coverage, function point testing, lines of code coverage and mission critical priority-based testing. McCabe's complexity measure will also be studied.
- Dynamic testing methods including unit, functional, big bang, system and regression testing. The regression testing component will also include a session on building and executing scripts.
- Debugging, including the use of assertions, validating contracts and debugging tools.
- Preventative measures including defect reduction techniques, source code metrics, best practices and continuous improvement techniques. Halstead will be studied along with maintainability measures and factors.

6. Future Work

In the proposed course, students will evaluate the tools and propose modifications and improvements. It is anticipated that MS projects or theses, undergraduate research opportunities or independent projects will be launched to implement the proposed improvements.

The authors intend to publish a follow-on paper documenting lessons learned and an evaluation of the tools used. The expectation is that this paper may become a model for other departments wishing to develop a similar course.

Data on student development efforts will be collected to determine if the course provides students with a functional basis to build "better" software. Possible study areas include code complexity, maintainability and improved skills in finding and preventing defects.

Finally, a proposal to develop a Software Quality Improvement Lab (SQIL) is under development. The proposal identifies a college-based laboratory dedicated to improving the quality of software. Industry partnerships are a critical component of the proposal, with the expectation that real-world development efforts may eventually be used as test targets within the lab.

7. Summary

Debugging and testing, and in general, software quality issues are not given adequate emphasis in a typical computer science curriculum. At the same time, software has become very pervasive and grown in complexity. Software developers are facing the growing challenge of testing these complex systems adequately. Our expectation is that by offering this course, some of our computer science graduates will acquire these skills and recognize the complexity and importance of debugging and testing software in a systematic way using appropriate tools.

References

[1] Barbosa, E. F., Maldonado, J. C., LeBlanc, R., & Guzdial, M. "Introducing Testing Practices into Objects and Design Course." *Proceedings of the 16th Conference on Software Engineering Education and Training,* (2003): 279-286.

[2] Carrington, D. "Teaching Software Testing." *Proceedings of the 2nd Australasian Conference on Computer Science Education,* (1997): 59-64.

[3] Computing Curricula 2001. Retrieved July 10, 2005, from the ACM Web site: http://www.sigcse.org/cc2001/

[4] Computing Curricula 2005: The Overview Report. Retrieved July 10, 2005, from the ACM Web site: http://campus.acm.org/public/comments/Draft_5-23-05.pdf
[5] Harrold, M. J. "Testing: A Roadmap." *Proceedings of the Conference on the Future of Software Engineering,* (2000): 61-72.
[6] Jones. E. "SPRAE: A Framework for Teaching Software Testing in the Undergraduate Curriculum." Retrieved July 10, 2005, from http://serel.cis.famu.edu/~testlab/Papers/SPRAE-framework.pdf
[7] Jones. E. L. "Software Testing in the Computer Science Curriculum: A Holistic Approach. *Proceedings of the 2nd Australasian Conference on Computer Science Education,*(2000): 153-157.
[8] Jones, E. L., & Chatmon, C. L. "A Perspective on Teaching Software Testing." *Journal of Computing Sciences in Colleges,*(March 2001): 16(3), 92-100.
[9] Kaner, C. "Teaching the Software Testing Course: A Tutorial." *Proceedings of the 17th Conference on Software Engineering Education and Training,* (2004).
[10] Kessler, M. "Fewer College Students Choose Computer Majors." Retrieved July 14, 2005, from the *USA Today* Web site: http://www.usatoday.com/tech/news/2004-08-08-computer-science_x.htm
[11] Marrero, W., & Settle, A. "Testing First: Emphasizing Testing in Early Programming Courses." *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, (2005): 4-8.
[12] McAllister, A., & Misra, S. "A Teaching Approach for Software Testing." *World Transactions on Engineering and Technology Education*, (2002): 1(2), 177-184.
[13] NIST Report on Software. Retrieved July 17, 2005, from the National Institute of Standards and Technology Web site: http://www.nist.gov/director/prog-ofc/report02-3.pdf
[14] Osterweil, L. J., et al. "Strategic Directions in Software Quality." *ACM Computing Surveys*, (1996): (4), 738-750.
[15] Pettichord, B. "Four Schools of Software Testing." Retrieved July 15, 2005, http://www.pnsqc.org/files /FourSchoolsofSoftwareTesting.pdf
[16] Tan, G. "A Collection of Well-Known Software Failures. " Retrieved July 18, 2005, from http://ww.cs.princeton.edu/~gtan/bug /softwarebug.html
[17] Thomas, C. "Hit the Hot Button for Jobs. " Retrieved July 15, 2005, from http://www.graduatingengineer.com/futuredisc /compsci.html