Software Testing in the Computer Science Curriculum -- A Holistic Approach

Edward L. Jones
Florida A&M University
Tallahassee, FL USA 32307
ejones@cis.famu.edu

Abstract

Although testing accounts for 50% of the cost of software, it receives little treatment in most curricula. This paper presents some approaches to giving all students multiple, incremental exposures to software testing throughout the curriculum. A unifying framework is presented which identifies a minimal set of test experiences, skills and concepts students should accumulate. The holistic approach combines common test experiences in core courses, an elective course in software testing, and volunteer participation in a test laboratory.

1 Introduction

Software testing is a major challenge in industry, accounting for 50% of the software costs. As software complexity and legal ramifications of poor quality increase, software testing will play an even more prominent role[9]. In the remainder of this section we assess the state of practice in software testing, and review approaches taken by educators to address the software quality issue.

1.1 State of Practice

There are many reasons to test. The classical statements of purpose by Dijkstra[2], "testing demonstrates the presence of bugs, not their absence," and Myers[8], "testing is the process of executing a program with the intent of finding an error," belie the diversity of motivations for testing. Table 1 depicts three dimensions of the software testing problem: purpose, granularity and strategy. Other dimensions exist, e.g., technology/architecture, or the target environment [8]. This characterisation shows the breadth and depth of testing concerns.

Table 1. Three Dimensions of Software Testing

Purpose	Granularity	Strategy
Acceptance	Function	Specification Driven
Defect	Object	*(black box)*
Reliability	Component	Implementation Driven
Performance	Application	*(white-box)*
Usability	System	Usage Driven
		(operational)
		Random *(statistical)*
		Intuitive *(ad hoc)*

The definitive work on software testing is Glenford Myers' book [8], *The Art of Software Testing*. Myers identifies attitude (psychology) and economics as major obstacles to testing. His oft-forgotten principles should be posted as the 13 Commandments of Testing Practice. Besides attitude, testers must also possess superior technical skills. Whittaker [11] points out the need for testers to have technical sophistication:

> "Testers must not only have good development skills--testing often requires a great deal of coding--but also be knowledgeable in formal languages, graph theory, and algorithms."

Technical approaches to testing have been published widely but, according to Osterweil [9], there is

> "... a yawning chasm separating practice from research that blocks needed improvements in both communities."

Three parties have a role to play to spanning this chasm. Companies must recognize that testing is a hard problem that is becoming harder as software becomes more complex. Researchers must produce approaches that are less academic and more practical (palatable) to industry consumers [11]. Finally, educators must do a better job equipping students with skills and attitudes for dealing effectively with software quality concerns.

1.2 Testing in the Curriculum

Software testing is ostensibly absent from computer science curricula. A premise of this work is that students who learn to test become better software developers because testing forces the integration of concepts and practice. Students who have been exposed to testing enter the workforce better equipped to deal with the *engineering* aspect of software.

Although testing has been a part of classical computer science literature for decades [8], the subject matter is seldom incorporated into the mainstream undergraduate experience. That so few articles on teaching software testing have been published suggests that software testing is an afterthought. In the article "Software Quality: A Curriculum Postscript?" Hilburn[3] makes a case for a comprehensive, end-to-end treatment of software quality by employing Watts Humphrey's team software process (TSP) and personal software process (PSP) throughout the curriculum [4,5].

Carrington [1] describes a testing course intended to give students practical skills along with theoretical knowledge of testing techniques. Students gain experience deriving test cases from formal specifications (black-box) and from code (white-box). Carrington reports that students come face to face with the labor-intensive drudgery of testing, and see firsthand the need to automate mundane tasks using scripting.

Jones [6] presents the SPRAE framework that identifies essential testing principles students should learn. The SPRAE principles include:

- *Specification.* A specification is essential to testing.
- *Premeditation.* Testing requires a systematic process.
- *Repeatability.* The testing process and its results must be repeatable and independent of the tester.
- *Accountability.* The test process must produce an audit trail.
- *Economy.* Testing must not require excessive human or computer resources.

These principles are used to design a set of test experiences students need to acquire.

1.3 Quality Emphasis in the Curriculum

Much has been published about computer-based evaluation of student work. When programs are being evaluated, the evaluation process is an instance of the testing problem (or software quality in the broader sense). For most students, this is the only time quality becomes important, because their grade is at stake. Our holistic approach capitalizes on program grading events as opportunities to expose students to software testing.

The work of Jones [7] and Reek [10] take a tester's approach to grading student programs. The assignment specification is the basis for test case design. Execution of programs against the test cases is automated using test harnessing techniques. Students learn quickly the most important lesson in software quality, that the specification is king! Failure to satisfy the specification *is* failure.

2 The Holistic Approach

It is impractical to teach every student all there is to know about testing. It is possible, and desirable, however, to give every student a common set of testing experiences that apply essential principles inherent in the practice of quality assurance. We believe the SPRAE framework identifies these essential principles, and that experience applying these principles transfers to new test situations.

The holistic approach has the following objectives.

- Every student has multiple testing experiences.
- Each experience incorporates SPRAE principles.
- Each course provides at least one test experience.
- Students are afforded opportunities for advanced experiences..

The holistic approach has the following components.

- The SPRAE framework defining essential principles and practices.
- Common testing experiences integrated throughout the curriculum.
- Testing methods applied to grading student programs.
- Elective course in software testing.
- Software test lab for advanced application of testing.
- Test lab products absorbed into the academic effort.
- Test lab provides testing services.

We believe that a holistic approach is more consistent with the way students learn life-long skills:

- SPRAE provides a small set of simple principles.
- Repetition of similar experiences occurs in different contexts (courses).
- Experiences are accumulated over two or three years.

3 Implications of the SPRAE Framework

In this section, we identify a minimal set of test experiences that cover the SPRAE-compliant test life cycle shown in Table 2. Student experiences begin at the end of the life cycle, and move backward toward the front as the student gains experience. A minimal sequence of test experiences is shown below.

1. Debug a program, given the test cases, and a test log.
2. Develop a test log from test results, given test cases.
3. Run tests, given test script and test data.
4. Design test cases, given a specification.

5. Refine a specification.
6. Develop a program, given its specification.
7. Create a test driver.

Recall the various dimensions of the software testing problem. The set of experiences listed above may be repeated for instances of the testing problem that occur in different courses. Each repetition reinforces techniques and principles learned earlier.

4 Elective Course in Software Testing

The first approach we attempted was to teach an elective course in software testing. The major benefit was that students received in-depth treatment of concepts and practices, preparing them for further study or application. The major disadvantage was that the course was not available to all students. Most students enrolled in the course only because they needed it to graduate, not because they were interested in testing. Regrettably, students who are genuinely interested in testing may not be able to take the course because it is not offered regularly.

Table 2. Test Life Cycle

Phase	Activity => *Products*
Analysis	Define test purpose and strategy. Refine specification. => *Test Plan + Refined Spec*
Design	Systematically derive test cases. Organize test effort. => *Test Cases + Test Data*
Implementation	Develop test "machinery". => *Test Harness + Test Scripts.*
Execution	Run test as a controlled experiment. Capture test results. => *Test Results*
Evaluation	Verify results. Take follow-up actions. Debug => *Test Log + Modified Software*

We gained important insights from teaching the course. First, we discovered that testing is a programming intensive activity. The students whose programming skills were rusty had a difficult time with the programming aspects of testing. We also discovered that testing requires analytical skills and facility with mathematical reasoning tools such as decision tables and graphs. Again, since these topics had been covered several semesters beforehand, students had forgotten. Students were shocked by the amount of record keeping testing requires. Their experience vacillated between boredom with record keeping, to extreme difficulty with the programming and analytical tasks.

Our conclusion is that a separate course in software testing should not be the only place to incorporate testing into the curriculum. Students who wait to take the elective course lose numerous opportunities to learn and reinforce testing skills in contexts other than "testing class." Students are notorious for compartmentalizing knowledge to a single course, and not transferring it to new situations. The preferred use of an elective course is to prepare students for advanced study in software testing.

5 Automated Program Grading

Our initial effort to incorporate testing into the curriculum was to automate the grading of student programs [7]. The automated program grading system (APGS) brings students face to face with quality assessment: poor quality results in a poor grade. Many students are surprised to learn that they really must follow the specification. This realization alone justifies the additional effort the teacher must invest to specify a testable assignment. Conscientious students soon begin to anticipate how their programs will be graded. That is, they do intuitive test case design based on the assignment specification. They begin to think like testers!

A SPRAE-compliant process is followed to transform an initial assignment specification into a *testable* specification. Each new assignment is added to a testbed of examples showing faculty (or teaching assistants) how to follow the process.

1. Develop initial assignment specification A0.
2. Transform A0 into a testable specification A1.
3. Design test cases from assignment specification A1.
4. Develop assignment grading plan P.
5. Design assignment grading scripts based on plan P.
6. Grade programs using grading scripts.
7. Distribute grading logs and scores to students.

The initial version of APGS is being used in an upper level elective course. We plan to move backwards through the curriculum, applying APGS to one additional course each year. We also plan to make the test design process and its artifacts more visible to students. Initially, we will make the test cases (for interactive programs, usage scenarios) available to students. Next, we will make the test plan available to students. Additionally, we will invoke the testing process at the time students submit their programs, as a way for the student to preview the quality of the program. The student will be permitted unlimited *dry runs* of submitted programs.

6 Experience Factory - The Software TestLab

The TestLab provides a SPRAE-compliant environment in which students can learn the art and science of software testing. TestLab students work on specific testing tasks to

produce and disseminate results in the form of completed artifacts, tutorials, experience reports and research publications. We seek to motivate students towards careers and graduate study in software testing, while transferring lessons and artifacts from the TestLab into courses in the undergraduate curriculum.

The TestLab is funded by corporate partners and by an NSF grant. The NSF support targets students who plan to attend graduate school. Corporate sponsorship extends student involvement to all students. Corporate partners are also encouraged to offer students internships in software testing/quality assurance.

TestLab students are required to take the elective course in software testing to acquire the background needed to be productive on TestLab assignments. We have formulated a training ladder reflecting the accumulation of testing skills and experiences from the course and TestLab. Table 3 describes the competencies TestLab students will acquire. Note that certain competencies focus on a specific phase of the test life cycle, while others (like Test Specialist) span multiple phases.

Table 3. TestLab Competency Categories

Competency	Duties
Test Practitioner	Perform a defined test within an established test environment and document results.
Test Analyst	Handle front-end of testing process. Determine testing needs. Assure that the specification is an adequate basis for starting the testing process.
Test Designer	Given a specification, design test cases using published and TestLab techniques.
Test Builder	Construct the machinery needed to run the test cases.
Test Inspector	Verify that a testing task has been performed correctly according to TestLab procedures and standards.
Test Environmentalist	Technician. Set up and support the test environment.
Test Specialist	Perform a series of testing tasks spanning multiple phases of the test life cycle.

The training ladder reflects progression through levels of competency within a category. Competency certification is earned by the completion of testing tasks representative of the skill level. As TestLab students progress, they become capable of performing tasks that support the dissemination mission of the TestLab. Competent TestLab students may be outsourced to faculty members to perform testing tasks such as automated grading or tutoring.

7 Conclusions

Although software testing is a difficult subject, students should acquire experience in fundamental aspects of testing. To date, our experience is that, when given in small doses, students absorb testing concepts and practices. Software testing forces students to integrate concepts, skills and practices. The holistic approach makes this integration a continuous process.

8 Future Work

This paper describes results from the first year of a five-year project. The work for the second year includes the following.

1. Develop web-based resources to support the software testing course. Provide ad hoc access to all students. Design course-specific entry points to support future courses.
2. Implement the TestLab experience ladder as a part of the TestLab management process. As the number of TestLab students increases, the experience ladder sets well-defined competency goals, and enables student-to-student mentoring.
3. Extend automated program grading to one additional course. Outsource competent TestLab students as technical support for refining assignment specifications and developing grading programs.

9 Acknowledgements

The author thanks the TestLab students who worked on ill-defined, yet groundbreaking tasks. These corporate sponsors provided moral and financial support: Lucent Technologies, Dell Computer, Lockheed Martin, 3M, and Abbott Laboratories. This work was partially funded by National Science Foundation grant EIA-9906590.

References

[1] Carrington, D. Teaching Software Testing. *Proceedings of Second Australasian Conference on Computer Science Education* (1996), 59-64.

[2] Dahl, O.J, Dijkstra, E.W., and Hoare, C.A.R. Structured Programming. APIC Studies in Data Processing 8. Academic Press, 1972.

[3] Hilburn, T.B., and Towhidnejad, M. Software Quality: A Curriculum Postscript? *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education* (May 2000), 167-171.

[4] Humphrey, W. S., *The Discipline of Software Engineering*, Addison-Wesley, 1995.

[5] Humphrey, W. S., *Introduction to the Team Software Process*, Addison-Wesley, 2000.

[6] Jones, E.L. The SPRAE Framework for Teaching Software Testing in the Undergraduate Curriculum. *Proceedings of Symposium on Computing at Minority Institutions*, (June 1-4, 2000).

[7] Jones, E.L. Grading Student Programs - A Software Testing Approach. *Journal of Computing in Small Colleges*, (November , 2000), to appear.

[8] Myers, G.J. *The Art of Software Testing*. John Wiley & Sons, New York, 1976.

[9] Osterweil, L., et al, Strategic Directions in Software Quality. *ACM Computing Surveys 28,4* (December 1996), 738-750.

[10] Reek, K.A. The TRY System – or – How to Avoid Testing Student Programs. *Proceedings SIGCSE Bulletin vol. 21*, 1 (February 1989), 112-116.

[11] Whittaker, J.A. What is software testing? And why is it so hard? *IEEE Software 17, 1* (Jan 2000), 70-79.