New Orleans JAZZED about Engineering Education

Paper ID #16425

Inclusive and Evidence-based Instruction in Software Testing Education

Vignesh Subbian, University of Cincinnati

Vignesh Subbian is an instructor in the Department of Electrical Engineering and Computing Systems at the University of Cincinnati. His interests include biomedical informatics, healthcare systems engineering, STEM integration, and engineering education.

Dr. Nan Niu, University of Cincinnati

Nan Niu is an Assistant Professor of Electrical Engineering and Computing Systems at the University of Cincinnati. He received his Ph.D. in Computer Science in 2009 from the University of Toronto, where he specialized in requirements engineering for software product lines. His current research interests include information seeking in software engineering, requirements engineering, program comprehension, and software engineering education. He is a member of ASEE and a senior member of IEEE.

Dr. Carla C. Purdy, University of Cincinnati

Carla Purdy is an associate professor in the School of Electrical Engineering and Computing Systems, College of Engineering and Applied Science, at the University of Cincinnati and an affiliate faculty member in UC's Department of Women's, Gender, and Sexuality Studies. She received her Ph.D. in Mathematics from the University of Illinois in 1975 and her PhD. in Computer Science from Texas A&M University in 1986. She is the head of UC's B.S. in Computer Engineering Program and the coordinator of the Preparing Future Faculty in Engineering Program. Her research interests include embedded systems and VLSI, intelligent embedded systems, software and systems engineering, computational biology and synthetic biology, agent based modeling and simulation, mentoring, and diversity in science and engineering.

Inclusive and Evidence-based Instruction in Software Testing Education

Abstract: This work-in-progress paper will present our experiences in developing a new Software Testing and Quality Assurance (QA) course that integrates evidence from research and new developments in software testing as well as engineering education. The specific goals are:

1. To incorporate empirical studies in software engineering to supplement instruction in testing of all aspects, including safety, security, reliability, and performance.
2. To increase focus on particular topics of high relevance such as formal testing of safety-critical systems and software inspection through targeted pedagogical interventions.
3. To leverage existing instructional materials from the software engineering education community to create and explore blended learning models such as a flipped classroom.
4. To integrate and promote inclusive and reflective teaching practices in computer engineering courses, in general.

We present detailed courseware and instructional modalities, including implementation details of daily in-class active learning activities, out-of-class assignments, and project resources, along with supporting materials from the literature and student feedback results.

1. Introduction

Evidence-based instruction or education is generally considered as the utilization of existing evidence from research and literature on education[1]. While findings from educational research are critical, appraisal of discipline-specific research is often overlooked or not well-integrated into instruction. We propose a unique research-to-practice model that combines evidence from research on education as well as the discipline itself. As a case in point, research literature in software testing is rich in empirical studies on software testing techniques, tools, and processes, encompassing systematic testing of both traditional stand-alone software systems as well as the increasingly larger systems of systems and embedded software[2]. Educators and educational researchers, on the other hand, have equally contributed towards pedagogical methods and instructional materials for software engineering education. In this paper, we present our experiences in developing a new Software Testing and Quality Assurance (QA) course that integrates evidence from research and new developments in software testing as well as engineering education.

2. Background

Our department is currently developing several new courses in an effort to expand the undergraduate and graduate curriculum in crosscutting areas of software engineering, embedded systems, and cybersecurity. The software engineering curriculum (Table I), in particular, has been expanded significantly during the 2014-15 academic year with the introduction of three new area-specific courses (CS 6027 – Requirements Engineering, CS 6028 – Large-scale

Software Engineering and EECE 6032 – Software Testing and Quality Assurance) and one cross disciplinary course (CS 7040 – Trustworthy System Design, Implementation, and Analysis). These new courses build upon the undergraduate introductory software engineering course[3] and complement the existing embedded systems curriculum[4,5].

Table I Core Software Engineering Courses

Course ID*	***Course Name (credit hours)***	***Level&***
EECE 3093C	Software Engineering (4)	U
CS 6027	Requirements Engineering (3)	U/G
CS 6028	Large-scale Software Engineering (3)	U/G
EECE 6032	Software Testing and Quality Assurance (4)	U/G
EECE 8XXX	Topics in Software Engineering (3)**	G

* 'C' denotes Integrated lab component; & U – Undergraduate, G – Graduate; ** under development;

3. Methods (Courseware)

The overall goal of the EECE 6032 – *Software Testing and Quality Assurance* course was for each student to understand the basic principles of software testing and quality, and their role in contemporary software engineering. An additional goal for graduate students was to examine research areas of interest, and be prepared to conduct research in software engineering in general. The ABET student learning outcomes of the course were:

- To understand how to develop a test plan for a set of software requirements and how to measure the quality of software and the development process itself (a, e)
- To comprehend the software testing and quality assurance processes for both traditional and distributed projects (a, g)
- To apply testing and quality assurance concepts to small-scale software projects (a, c, e, g, k)
- To comprehend formal verification methods (a, e)

The course was designed to include in-class learning through group problem-solving and traditional lectures, out-of-class learning through online lectures and/or research literature reading for selected topics, and a semester-long team project focused on application of testing techniques as well as performing QA activities. Additionally, graduate students were required to complete a research component.

Course Topics

The topics in the EECE 6032 course were primarily aligned with two Knowledge Areas (KAs) in the Software Engineering Body of Knowledge (SWEBOK): software testing (KA5) and software quality (KA10), with some overlap in other relevant KAs such as software

configuration management, software engineering process, and software engineering models and methods (see Table II for details).

Course Delivery

We used different instructional modalities and methods based on the nature and priority of topics. For example, traditional lectures on most testing techniques were followed by in-class group problem solving (marked as ✓ in Table II). These in-class activities were designed to immediately apply and reinforce concepts covered in the lecture.

Selected concepts such as test selection criteria, test oracles, exploratory testing, and the complexity (or perhaps, the impossibility) of complete/exhaustive testing, were offered as "blended learning (flipped)" modules i.e., students watched video lectures outside of class (marked as 💻 in Table II) and spent the class time for group problem-solving and discussion. Although there was no reduction in face-to-face (F2F) time, we refer the combination of F2F interactions and out-of-class online instruction as "blended learning". This was made possible by the availability of publicly-available instructional materials that were developed through federally funded projects for the sole purpose of improving education in software testing (see "Acknowledgment" section for details).

Integration of Evidence from Software Engineering Research

A number of empirical studies in the software engineering research literature were utilized to supplement instruction of several topics (marked as 🕮 in Table II) throughout the course. Such studies, particularly the results, were introduced as a part of in-class lectures or discussions. Students were then asked to critically review the study as an out-of-class assignment, reflect on how the concepts and results related to their current understanding, and report a summary and reflection, with particular emphasis on quantitative results. For example, to highlight the inefficacy of code coverage as a metric of test suite quality, we incorporated a recent study by Inozemtseva and Holmes[6]. A total of eight experimental studies[6-13], one case study[14], and two survey/review articles[15,16] were integrated into the course content. Of these, five studies/articles had accompanying required "review-reflect-and-report" assignments. It is noteworthy that reflection was a critical aspect of these assignments. Students were explicitly asked to retrospect on their prior experiences in software engineering projects and identify how it relates to concepts, results and/or arguments in their reading assignments.

Table II Course Schedule

I = In-class group activity; **O** = Online lecture; **E** = Integration of evidence from research literature			
Topic (hours*)	I	O	E
Software Testing Fundamentals (6)			
- Testing-related Terminology			
- Static versus Dynamic Testing			🕮 10
- Test Selection Criteria		💻	🕮 7
- Test Oracles		💻	
- Complexity of Exhaustive Testing		💻	🕮 8
Levels of Testing(6)			
- Unit Testing			
- Integration Testing	✓		
- System-level Testing			
Testing Techniques(13)			
- Coverage-based Testing	✓	💻	🕮 6
- Input Domain-based Testing	✓		🕮 8
- Control flow and Data flow Testing	✓		
- Mutation Testing	✓		🕮 9
- Usage-based Statistical Testing	✓		
- Model-based Testing			🕮 11
• Finite-state Machines			
• Testing from Formal Specifications			🕮 15
• Model Checking	✓		
- Exploratory Testing		💻	
*Testing Process (1) **			
- Test Activities			
Software Quality(8)			
- Relationship of Testing to Software Quality			
- Software Quality Assurance			
• Defect Prevention			
• Fault Tolerance and Failure Containment			🕮 12
- Software Quality Improvement			
- Software Process Assessment and Improvement (Maturity Models)	✓		
- Verification versus Validation			
- Software Quality Metrics			
Special Topics (6)			
- Software Configuration Management			
- Formal Verification of Safety-critical Systems			🕮 15,16
- Testing in Distributed/Global Software Development			🕮 13
- Intellectual Property and its Implications for Software Testing			

* Includes online lecture hours; ** Extensively covered as a part of the team project

Team Project and Research

Each student team consisted of 3-4 members with at least one graduate student and one student in the computer science program. The goal of the team project was to provide an opportunity for students to apply some specific testing techniques or tools to one or more chosen System Under Test(s) (SUTs) of interest (either open-source software, or software that they developed for other projects). The minimum project requirements were: (1) including both testing and QA components, although it was up to each team to decide on the proportion of both components, (2) developing and executing a test plan, even if testing was a small part of the project, and (3) performing a manual software inspection for selected modules or the whole SUT.

Students were encouraged to leverage publicly-available SUTs and test suites to apply testing techniques and tools. Here, we provide a summary of resources that other educators can use as well as provide as recommendations to their students.

- Software Assurance Reference Dataset (SARD)[17]: Developed and maintained by the National Institute of Standards and Technology (NIST), SARD provides a collection of test programs and test suites, along with known defects and vulnerabilities (most defects are security-related). The test cases in the dataset are well-documented and allows for understanding the defects as well as writing new test cases.

- Recommended open-source test suites and test programs: apache-poi[18], google-vis[19], jdom[20], jfreechart[21], jgrapht[22], jmeter[23], joda-time[24], and weka[25].

For the research component, graduate students were required to identify and answer a research question of interest. Undergraduate students shadowed graduate student mentors in their team and completed a short abstract along with a reflection of their experience. To allow for interactions with different programs within engineering and computer science, the research component was integrated into assignments, and the second offering (Fall 2015) increased focus on interdisciplinary team projects, including tool-supported test management and testing of large-scale software-intensive systems.

Targeted Pedagogical Interventions

In order to increase focus on topics of high relevance to testing of software-intensive safety-critical systems[26, 27] (formal verification and software inspection), two interventions were implemented:

1. Besides regular instruction and learning activities on formal methods, a leading researcher and proponent of formal methods was invited to construct and present a follow-up talk on formal verification of safety-critical systems. The goal of this intervention was to emphasize on construction of formal specifications for automatic

static analysis and symbolic model checking, both, in the context of testing software in aerospace systems.

2. For software inspection, students were required to perform manual inspection of their SUTs as a part of their team project. Additionally, an invited lecture was arranged to promote an awareness of intellectual property and its implications for software testing, particularly during manual or tool-supported inspection.

Inclusive Teaching Practices

As a part of our personal interest in creating inclusive learning experiences for engineering students, we adopted several strategies to uphold inclusivity in teaching and learning. We report these ideas and methods here for several reasons: (1) some of these strategies were inspired by diversity initiatives led by ASEE as well as our own institutional/departmental working groups and therefore, we believe that reporting our experiences will highlight and better capture the impact of such initiatives, (2) inclusiveness is presumably perceived and actualized in different ways in engineering education, and it may be important to identify and share the differences and similarities in inclusive practices and its significance in development of engineers, and (3) we believe this will stimulate educators to consider, develop, and report such practices as a part of their course or program development. Here, we summarize a few inclusive approaches as they relate to software engineering education.

- Developing and including a statement of inclusion in the syllabus such as, "The diversity of the participants and their ideas are a valuable source of ideas and *software engineering creativity*…", and mindfully putting the statement into practice throughout the course delivery and interactions with students.

- Featuring pioneers in software engineering during in-class discussions and presentations: Kent Beck and Eric Gamma (unit testing), Richard Battin and Margaret Hamilton (safety-critical systems), Martin Fowler (continuous integration, extreme programming), Grace Hopper (who coined the term software "bug" and "debugging"), John Musa (software reliability and usage-based statistical testing), and Mary Shaw (software architectures). Contributions of these pioneers and brief historical perspectives were integrated at appropriate places into course activities and materials.

- Ensuring diversity in reading materials (see assigned articles[6-13]) and guest speakers.

Assessment

Students were assessed based on the following grade distribution: In-class activities and participation (10%), out-of-class assignments (25%), two exams (15% each), testing and QA team project (35%).

4. Preliminary Results

The course has been offered twice so far (Spring 2015 and Fall 2015) serving a total of 46 students from three program areas (computer science, electrical engineering, computer engineering). Figure 1 and Figure 2 show demographics of course participants. While the course was open to both undergraduate seniors and graduate students, only the first offering had students from both levels (13 graduate and 9 undergraduate seniors); participants in the second offering were all graduate students. Table III shows qualitative feedback obtained from students on various course activities. The majority of students found the course to be well-structured and collaborative. The in-class activities, in particular, received positive reviews.

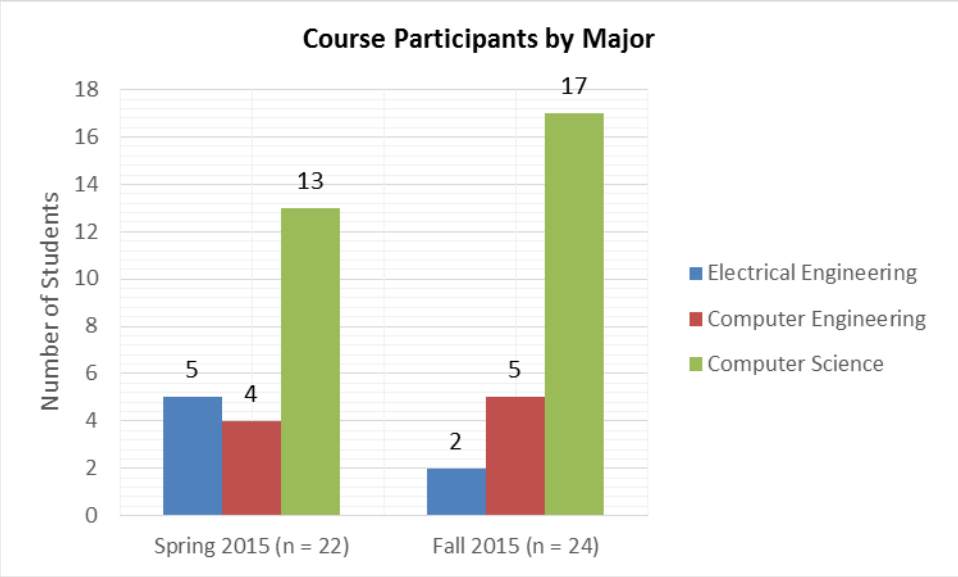

Figure 1 Enrollment Demographics of Course Participants by Major

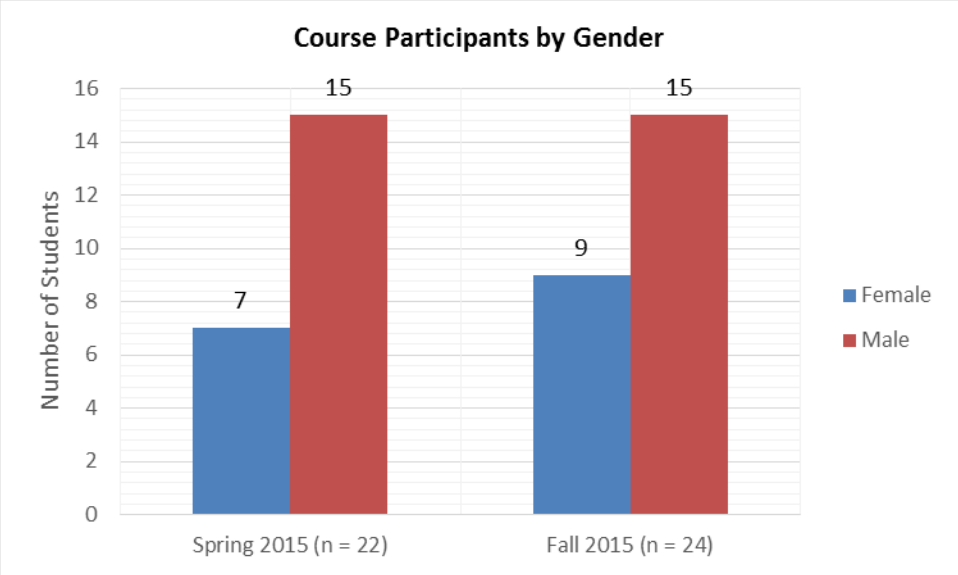

Figure 2 Course Participants by Gender

Table III Qualitative Student Feedback Results

Course activity	***Excerpts from student feedback***
In-class group activities	"One of my favorite parts were the in-class activities because it allowed me to apply the material that we just learned as well as work with other students and get to know them."
Intervention on formal methods	"The idea of formal methods and automated testing is absolutely amazing. After this talk, it definitely makes logical sense how it is done. All of this relates to my current work experience and I am eager to get back to work and apply them to a project that might not have these tools implemented yet!"
Blended learning	"This style of teaching is still relatively new to me. The videos that were posted articulate the material well. I think this course could be done with a reverse (flipped) class room setting if the size stays small."
	"Regarding the proposed new model of the class of watching some videos or reading some papers then discussing them in class, I do not really like it. I do find it useful to review the current literature and research on the topics, but I think it would be more worthwhile to watch the videos or read the articles in class and discuss them as a whole."

Limitations and Future Work

First, there were no quantitative methods or a control group to statistically measure the impact of pedagogical interventions. However, based on the qualitative results, we believe that our implementation was successful, particularly in emphasizing and promoting formal methods and software inspection. Furthermore, the integrated inspection component in the team project and the two invited in-house experts are sustainable resources for subsequent offerings of this course.

Second, tool-supported test plan management was highly encouraged, but not mandated in team projects. Streamlining tool usage for test management as well as static and dynamic code analysis are a part of ongoing work. Last, blending learning was limited to a few topics and does not represent the true scope of the technique. We are currently developing an online version of the course and exploring how we can leverage the online content for our on-campus students.

5. Conclusion

In summary, we have successfully developed and implemented a new course in software testing and quality assurance that integrates blended learning, evidence from software engineering research, and topic-specific interventions. We believe that this work will be of interest to practitioners of evidence-based instruction and other educators in the software engineering community.

Acknowledgment

We acknowledge the use of video lectures created by Prof. C. Kaner (Director, Center for Software Testing Education and Research, Florida Institute of Technology) and team. These video lectures were used for implementing blended learning modules in our course. We thank Prof. T. Armstrong (College of Law, University of Cincinnat), Prof. K. Rozier (College of Engineering and Applied Science, University of Cincinnati and Mr. C. St. Pierre (AppDynamics Inc.) for their contributions to this course.

References:

1. Davies P. What is evidence- based education? *British journal of educational studies*. 1999;47(2):108-121.

2. Ostrand T, Weyuker E. Software testing research and software engineering education. *Proceedings of the FSE/SDP workshop on Future of software engineering research*. 2010:273-276.

3. Subbian V, Purdy C. A hybrid design methodology for an introductory software engineering course with integrated mobile application development. *Annual ASEE Conference*. 2014.

4. Subbian V, Purdy C. Redesigning an advanced embedded systems course: A step towards interdisciplinary engineering education. *IEEE Integrated STEM Education Conference*. 2013.

5. Subbian V, Beyette F. Developing a new advanced microcontrollers course as a part of embedded systems curriculum. *Frontiers in Education Conference*. 2013:1462-1464.

6. Inozemtseva L, Holmes R. Coverage is not strongly correlated with test suite effectiveness. *Proceedings of the International Conference on Software Engineering*. 2014:435-445.

7. Harder M, Morse B, Ernst MD. Specification coverage as a measure of test suite quality. *ACM*. 2001;25:452.

8. Kuhn DR, Wallace DR, AM Gallo J. Software fault interactions and implications for software testing. *Software Engineering, IEEE Transactions on*. 2004;30(6):418-421.

9. Just R, Jalali D, Inozemtseva L, Ernst MD, Holmes R, Fraser G. Are mutants a valid substitute for real faults in software testing. *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 2014.

10. Ayewah N, Pugh W, Morgenthaler JD, Penix J, Zhou Y. Evaluating static analysis defect warnings on production software. *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. ACM. 2007:1-8.

11. Dalal SR, Jain A, Karunanithi N, et al. Model-based testing in practice. *Proceedings of the 21st international conference on Software engineering*. 1999:285-294.

12. Knight JC, Leveson NG. An experimental evaluation of the assumption of independence in multiversion programming. *Software Engineering, IEEE Transactions on*. 1986(1):96-109.

13. Bird C, Nagappan N, Devanbu P, Gall H, Murphy B. Does distributed development affect software quality?: An empirical case study of windows vista. *Commun ACM*. 2009;52(8):85-93.

14. Jee E, Wang S, Kim JK, Lee J, Sokolsky O, Lee I. A safety-assured development approach for real-time software. *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. 2010:133-142.

15. D'silva V, Kroening D, Weissenbacher G. A survey of automated techniques for formal software verification. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*. 2008;27(7):1165-1178.

16. Sandler K, Ohrstrom L, Moy L, McVay R. Killed by code: Software transparency in implantable medical devices. *Software Freedom Law Center*. 2010:308-319.

17. National Institute of Standards and Technology. Software assurance reference dataset (SARD). https://samate.nist.gov/SARD/. Updated 2015.

18. Apache POI. Apache-poi. https://github.com/apache/poi. Updated 2016.

19. google-vis. Google visualization. https://github.com/google/google-visualization-java.git. Updated 2012.

20. JDom. https://github.com/hunterhacker/jdom. Updated 2015.

21. Gilbert D. jFreeChart. https://github.com/jfree/jfreechart-fse. Updated 2016.

22. Naveh B. JGraphT. https://github.com/jgrapht/jgrapht. Updated 2016.

23. Apache JMeter. JMeter. https://github.com/apache/jmeter. Updated 2016.

24. Joda-time. https://github.com/JodaOrg/joda-time. Updated 2015.

25. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The WEKA data mining software: An update. https://github.com/bnjmn/weka. Updated 2009. Accessed 1, 11.

26. Lutz RR. Software engineering for safety: A roadmap. *Proceedings of the Conference on the Future of Software Engineering*. 2000:213-226.

27. Parnas DL, van Schouwen AJ, Kwan SP. Evaluation of safety-critical software. *Commun ACM*. 1990;33(6):636-648.