WIP: Active Learning Exercises to Promote System Performance Testing

Dr. Walter W Schilling Jr., Milwaukee School of Engineering

Walter Schilling is a Professor in the Software Engineering program at the Milwaukee School of Engineering in Milwaukee, Wisconsin. He received his B.S.E.E. from Ohio Northern University and M.S. and Ph.D. from the University of Toledo. He worked for Ford Motor Company and Visteon Corporation as an Embedded Software Engineer for several years prior to returning for doctoral work. He has spent time at NASA Glenn Research Center in Cleveland, Ohio, and consulted for multiple embedded systems software companies in the Midwest.

In addition to one U.S. patent, Schilling has numerous publications in refereed international conferences and other journals. He received the Ohio Space Grant Consortium Doctoral Fellowship and has received awards from the IEEE Southeastern Michigan and IEEE Toledo Sections. He is a member of IEEE, IEEE Computer Society and ASEE. At MSOE, he coordinates courses in computer organization, secure software development practices, network security, software verification, real time systems, and operating systems, as well as teaching embedded systems software development.

Dr. Brad Dennis, Milwaukee School of Engineering

Dr. Brad Dennis is a new Software Engineering faculty member at the Milwaukee School of Engineering. He received is PhD. from Auburn University where also received his Master of Software Engineering and Bachelor of Software Engineering. He has over 20 years of experience in a diverse set of industries, including defense, healthcare, and most recently, professional training for network engineers.

©American Society for Engineering Education, 2017

WIP: Active Learning Exercises to Promote System Performance Testing

Abstract

The verification of system performance is a major aspect necessary to ensure the proper operation of software systems. In numerous, high-profile cases, deployed systems, such as e-commerce sites and the Healthcare.gov website, have failed due to performance related issues. The IEEE Software Engineering SE2014 document encourages software engineering programs to cover multiple forms of testing, ranging from unit tests through performance tests. Unit testing, at one extreme of this range, is easily taught as it focuses on small scopes and detailed functionality. Another type of testing, integration tests, can also be easily expressed based upon sequence diagrams. These two items must have deep coverage per the curriculum guidelines. However, at the other end of this spectrum is performance testing. Performance testing is much harder to teach, while also needing much less coverage per the IEEE SE2014 guidelines. This article will define a set of active learning exercises which were developed to aid in the understanding of performance testing. These exercises allow students to execute performance tests and analyze the results, while not requiring students to develop any performance tests. In each case, the performance behaviors are readily traced back to design and implementation decisions made which significantly limit the scalability of the systems under test. This article will describe our experiences with implementing the exercises, as well as provide feedback from students who have used the module to learn about performance testing.

Introduction

Verification is a significant skill that software engineers must master to be effective practitioners in the field. Numerous cases of software failure can be traced to a lack of appropriate Software Verification and Validation activities, ranging from the failure of the Healthcare.gov website [1] to the problems of unintended acceleration in Toyota vehicles [2] to the Heartbleed security vulnerability [3] to a ten-hour outage of the electronic medical records system at Queen Elizabeth Hospital [4].

The need for software engineers to be trained in the field of testing is well documented. Lethbridge [5] indicates that software testing and quality assurance is one of the more important topics for universities to include in their curriculum. However, his work also notes that this is one area in which on the job training often occurs because students are not taught adequate testing skills. In general, there is a shortage of trained practitioners who understand Software Verification and Validation activities and processes [6].

The IEEE Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering indicates that Software Verification and Validation are significant topics of coverage, with 37 hours of coverage recommended in the minimal curriculum covering the topics shown in Figure 1. Aside from basic computing essentials and mathematical foundations, Verification and Validation activities represent one of the larger curricular components for a recommended software engineering curriculum.

Reference	Topic	k,c,a	E,D	Hours
VAV	Software verification and validation			37
VAV.fnd	V&V terminology and foundations			5
VAV.fnd.1	V&V objectives and constraints	k	E	
VAV.fnd.2	Planning the V&V effort	k	E	
VAV.fnd.3	Documenting V&V strategy, including tests and other artifacts	a	E	
VAV.fnd.4)	Metrics and measurement (e.g., reliability, usability, and performance)	k	E	
VAV.fnd.	5 V&V involvement at different points in the lifecycle	k	E	
VAV.rev	Reviews and static analysis			9
VAV.rev.1	Personal reviews (design, code, etc.)	a	E	
VAV.rev.2	Peer reviews (inspections, walkthroughs, etc.)	a	E	
VAV.rev.3	Static analysis (common defect detection, checking against formal specifications, etc.)	a	E	
VAV.tst	Testing			18
VAV.tst.1	Unit testing and test-driven development	a	E	
VAV.tst.2	Exception handling (testing edge cases and boundary conditions)	a	E	
VAV.tst.3	Coverage analysis and structure-based testing	a	E	
VAV.tst.4	Black box functional testing techniques	a	E	
VAV.tst.5	Integration testing	c	E	
VAV.tst.6	Developing test cases based on use cases and/or user stories	a	E	
VAV.tst.7	Testing based on operational profiles (e.g., most-used operations first)	k	E	
VAV.tst.8	System and acceptance testing	a	E	
VAV.tst.9	Testing across quality attributes (e.g., usability, security, compatibility, and accessibility)	a	E	
VAV.tst.10	Regression testing	c	E	
VAV.tst.11	Testing tools and automation	a	E	
VAV.tst.12	User interface testing	k	E	
VAV.tst.13	Usability testing	a	E	
VAV.tst.14	Performance testing	k	E	
VAV.par	Problem analysis and reporting			5
VAV.par.1	Analyzing failure reports	c	E	
VAV.par.2	Debugging and fault isolation techniques	a	E	
VAV.par.	Defect analysis (e.g., identifying product or process root 3 cause for critical defect injection or late detection)	k	E	
VAV.par.4	Problem tracking	c	E	

- Knowledge (k): Remembering material learned previously. Test observation and recall of information; that is, "bring to mind the appropriate information" (such as dates, events, places, knowledge of major ideas, and mastery of subject matter).
- Comprehension (c): Understanding information and the meaning of material presented. For example, being able to translate knowledge to a new context, interpret facts, compare, contrast, order, group, infer causes, predict consequences, and so forth.
- Application (a): Using learned material in new and concrete situations. For example, using information, methods, concepts, and theories to solve problems requiring the skills or knowledge presented.

A topic's relevance to the core is designated in a similar manner:

- Essential (E): The topic is part of the core.
- Desirable (D): The topic is not part of the core, but it should be included in the core of a particular program if possible; otherwise, it should be considered part of elective materials.

Figure 1: IEEE SE2014 Software Verification and Validation Coverage [7]

One of the challenges of teaching software verification has been the lack of quality teaching materials at the undergraduate level. Through the years, there have been several projects aimed at creating case study modules for teaching. Most recently, the Software Development Case Study [8] project developed a set of case studies that can be used across the software engineering curriculum based upon the digital home. However, while testing materials were part of the project, the materials were not focused specifically on verification and validation.

A later NSF project, Collaborative Education: Building a Skilled Software Verification and Validation User Community [9], focused on developing active learning exercises for software

engineering. This paper provides initial feedback on the usage of one such activity in a software engineering program.

About the institution

The Milwaukee School of Engineering offers an accredited Bachelors of Science degree in software engineering, and has been accredited since 2002. As an institution, there is a strong emphasis on small class sizes (14:1 student to faculty ratio) and extensive laboratory experience. Students graduating from MSOE spend on average 600 hours in laboratories related to their major. Institutionally, there is more square footage devoted to lab space than lecture hall space. All engineering students are required to complete a three-course capstone experience. While most students on campus are in the engineering fields, the school also offers a nursing program, a user experience program, and several business programs. MSOE prides itself in having very few traditional computer labs on campus. Instead, all students enrolled in the university are issued a laptop as part of a technology package which includes the laptop and all relevant software needed for the program the student is enrolled in.

The software engineering program offers students several unique learning opportunities. One part of the program is a 12 credit Software Development Laboratory experience where students work on large-scale, industry-sponsored projects. Most core software engineering courses are offered in the 3+2 format, meaning the course meets in lecture three times for one hour and has a 2 hour associated lab period. Elective courses tend to be offered in the 2+2 format, with 2 hours of lecture instead of 3 and a 2 hour lab.

About the course

SE 3800 is a course designed to provide more in-depth discussion into the areas of agile software development and quality assurance processes. Topics included in the course are continuous integration, agile software development, and software quality assurance activities. It serves as a follow-on course to an introductory course in Software Engineering Processes as well as a follow-on to a course in Software Verification.

The first software engineering process course (SE2800 Software Engineering Process 1) serves to introduce students to agile development, including the software development lifecycle, effort tracking, project planning, measurement and estimation, and the SCRUM process. Specific outcomes for the course include the ability to:

- Understand basic concepts of software engineering process
- Understand software process and product metrics
- Work within a standard development process
- Document process and product measurements
- Plan and track software projects

The software verification course (SE2832 Introduction to Software Verification) introduces students to the fundamental concepts of software verification. Topics include an overview of basic testing, coverage criteria, basic testing metrics, and the application of basic testing tools. Specific course outcomes include the ability to:

- Explain why testing is critical to software development

- Explain the relationship between verification and validation
- Compose accurate and detailed defect reports and record defects into a defect tracking system
- Using appropriate coverage criteria and testing theory, design and construct high-quality testing approaches and prepare tests in a logical, organized fashion
- Apply testing theory to design tests based on presented test criteria
- Analyze the effectiveness of testing using testing metrics, mutation testing, and other techniques
- Design and implement test cases using Mock objects
- Analyze a given piece of source code for complexity and testability

About the Exercise

Performance testing is an important aspect of modern software development. However, in addition to measuring performance, it is also important that students understand the root cause for a given systems performance. Students also need to be able to determine the root cause of the problems as well as propose potential solutions.

Performance testing, however, by its very nature, can be complex to setup and execute. Moreover, while an understanding of performance testing is required by the software engineering curriculum guidelines, students are not expected to be able to implement or design a performance testing strategy.

As part of the Collaborative Education: Building a Skilled Software Verification and Validation User Community [9], an exercise was developed to help students understand performance testing. The exercise included three implementations of a simple web application. The web application allowed a user to enter a word. The application then searched through a set of words that were preloaded to find the word and list synonyms for the word. Two of the three implementations suffered from fundamental implementation flaws which would preclude the software's being deployed due to performance problems.

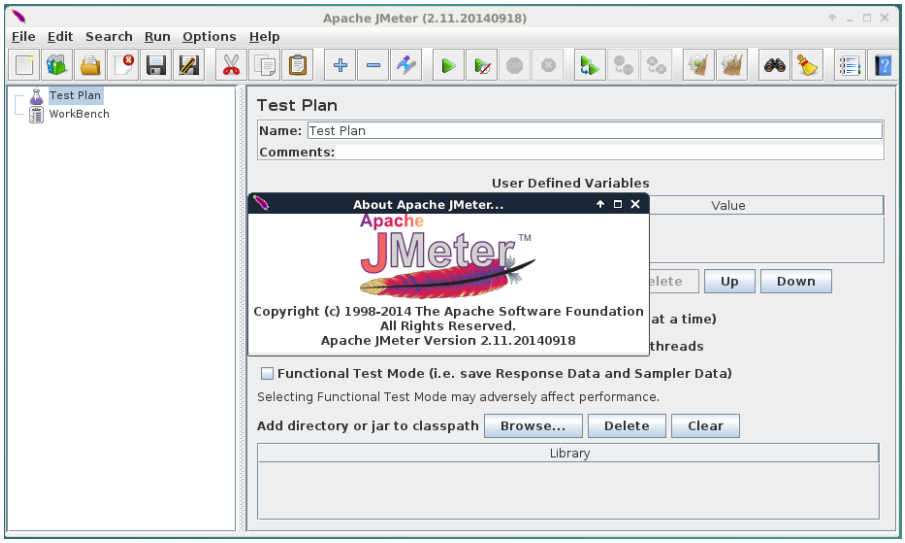

Figure 2: JMeter GUI User interface.

Students were provided with a set of JMeter test cases that were to be executed against each of the three implementations. Apache JMeter is a tool included within the Apache project designed for performing load testing on a variety of applications. The tool can be used as a load testing tool for analyzing and measuring the performance of a variety of services, though the focus is on web applications. Jmeter provides the ability for a user to use variable parameterization, response validation through assertions, thread specific cookies, dynamically configure behavior, as well as generate a series of detailed performance reports. JMeter also supports extension through the usage of plugins, though this exercise did not use this feature.

The applications and JMeter test cases were distributed as part of a VirtualBox Virtual Machine image that could easily be played back by the students. Based on the results of these tests, the following three goals were to be achieved:

- Determine which implementation performs better
- Determine which application will scale better to many users
- Using the implementation, find out why there is a difference in performance between the three systems.

The exercise was conducted with a lab partner. The active learning exercise used a set of prepared lecture slides from the project. However, extensive lecturing before the activity did not occur. Based on this limited introduction and previous experiences, the teams were tasked with answering the following questions:

1. *Which application is faster?*
2. *Which application delivers more throughput?*
3. *Discuss the performance differences between the three applications.*
4. *Which of the graphs stand out as being better or worse than the others regarding the deviation in response time?*
5. *Discuss any differences you identify.*
6. *Is there a relationship between the source code and the response times shown on the graphs?*
7. *Discuss your answer to the previous question.*
8. *Does it make sense why one application performs better or worse than the other applications based on the test results and source code?*
9. *Discussion 4. Which application performs the best?*
10. *Discuss the results from the previous two questions.*
11. *Is one application more scalable than the others, and thus, more suitable for deployment?*
12. *Which application is the most scalable?*
13. *Discuss the results from the previous two questions.*
14. *Does editing the word list CSV file to change the words have any impact on the performance of the system?*
15. *Discuss your answer to the previous question.*
16. *Does changing the email, the first or the last name, change the performance of the system?*
17. *Discuss your answer to the previous question.*
18. *What happens if the number of threads is increased?*
19. *Does this behavior make sense?*
20. *Discuss why the behavior does or does not make sense?*
21. *What happens if you change the loop count?*
22. *Does changing the loop count impact the performance of the system at all?*
23. *Discuss why or why not does the behavior make sense?*
24. *What happens as you run a list of misspelled words through the three different implementations?*

Analysis and Results

With the students completing the assignment, the goal of this analysis was to determine to what cognitive level students understood the concepts of performance testing and whether they could extrapolate and make appropriate conclusions based on test execution results. To do this, questions were grouped together based on the cognitive understanding of the results necessary and the conclusions drawn based on test results. To ensure the unbiased nature of the analysis, the instructor performing the evaluation was independent and was not involved in teaching the course.

The first question to be answered was whether the students were capable of properly reading the output of the tool. The tool (JMeter) provided a graphical output to students, such as is shown in Figure 3. The first questions merely involved students interpreting the graph, as the average time of the applications and throughput was readily plotted on each of the three applications. Student responses to the three questions were analyzed based on a four-point rubric. Responses and rubric entries are shown in Table 1.

Figure 3: Sample JMeter output

Virtually all students performed acceptably on this aspect of the assignment. Few students had a perfect analysis, but most students could answer this set of simple graph interpretation questions correctly. This activity, however, reflects a very low-level activity on Bloom's Taxonomy, simply applying graph interpretation skills to performance testing.

Table 1: Analysis of the first three questions.

- Which application is faster?
- Which application delivers more throughput?
- Discuss the performance differences between the three applications.

Rubric Score	Definition	Percent Achieving
4	**Accomplished:** Complete discussion of why the machines are different in performance. Correct identification of which machine was the fastest and which had the best throughput.	29%
3	**Proficient:** Correctly identified which machine had the highest throughput and which machine performed the best. Analysis and / or discussion incomplete.	64%
2	**Slightly Flawed:** Incorrect identification of machines with the best performance. Analysis complete but based upon flawed assumption.	7%
1	**Significantly Flawed:** Incorrect machine identified and analysis significantly flawed.	0%
0	No response	0%

The analysis of questions 4 and 5 also served to evaluate a low-level Blooms Taxonomy exercise. Students were asked to indicate which graph had the highest deviation in performance. This again involved simply reading the raw output of the tool. A similar rubric for assessment was used, and the results are shown in Table 2.

Table 2: Analysis of the second two questions.

- Which of the graphs stand out as being better or worse than the others regarding the deviation in response time?
- Discuss any differences you identify.

Rubric Score	Definition	Percent Achieving
4	**Accomplished:** Correct response as to which graph stands out. Complete and full analysis of why the graph is different. Explanation includes discussion of why the finding is relevant.	0%
3	**Proficient:** Students correctly identify a graph as having different amounts of deviation, but the differences are purely based on the number shown.	71%
2	**Slightly Flawed:** Students identified the difference, but the explanation did not involve any discussion of the graph.	21%
1	**Significantly Flawed:** The answer is clearly wrong, including misinterpretation of numbers and or application to the wrong test results.	7%
0	No response	0%

This problem, while seemingly just as simple as the first problem in that data was available on the graph, led to many teams being challenged. While the majority were still proficient, a significant percent did provide flawed answers. In the case of the significantly flawed responses, students misinterpreted the graphs and indicated the wrong implementation had the most deviation. Those that were slightly flawed tended to provide a somewhat correct answer but did not give a correct analysis as to why they selected the implementation they chose. Some provided no reasoning, while others provided reasoning that would not have been discernable from interpreting the tool outputs, discussing differences in source implementation instead.

The third question set increased the Bloom's Taxonomy level, requiring students to evaluate the performance behaviors seen by looking at the source code. The source code was developed in

Tomcat as a Java application, which had been taught in the web applications course, and the data structures used had been studied extensively in the data structures course taken previously.

Table 3: Analysis of the third two questions.

- Is there a relationship between the source code and the response times shown on the graphs?
- Discuss your answer to the previous question.

Rubric Score	Definition	Percent Achieving
4	**Accomplished:** A complete and full explanation of the justification for the performance difference in the source code. A correct indication that there is a solid difference due to implementation.	7%
3	**Proficient:** Correct indication that there is a difference in performance due to implementation. The answer indicates that the student understands why there is a difference but is not complete.	64%
2	**Slightly Flawed:** Correct identification of difference caused by source code. Analysis lacking or incorrect.	27%
1	**Significantly Flawed:** Did not correctly identify difference due to the source code. Analysis incorrect.	0%
0	No response	0%

The fourth set of questions involved whether the students could explain, in algorithmic terms, why the performance was the way that it was in testing. For this question series, students were expected to discuss the impacts of Iterators, Big-O notation, and elementary Java data structures (Hashmap, TreeMap, etc.). This answer showed that students seemed to lack an ability to explain the differences in performance using terminology that would reflect an understanding of data structures and algorithms in performance based terminology.

Table 4: Analysis of the fourth three questions.

- Does it make sense why one application performs better or worse than the other applications based on the test results and source code?
- Discussion 4. Which application performs the best?
- Discuss the results from the previous two questions.

Rubric Score	Definition	Percent Achieving
4	**Accomplished:** The answer correctly identifies the application which performs differently. The answer properly identifies within the implementation the root cause for the performance difference and the discussion includes algorithmic details which would explain the performance difference.	29%
3	**Proficient:** The answer correctly identifies the application which performs differently. The analysis is correct, but lacks algorithmic details justifying the performance.	36%
2	**Slightly Flawed:** The answer correctly identifies the application which performed best, but does not provide any evidence or analysis to support the answer.	29%
1	**Significantly Flawed:** The answer is incorrect or lacks any substantive explanation for the behavior that is seen.	7%
0	No response	0%

The fifth analysis of the students' performance looked at whether they could identify the scalable application. Students were asked which application was most scalable and then expected to provide justification for their answer.

Table 5: Analysis of the fifth three questions.
- Is one application more scalable than the others, and thus, more suitable for deployment?
- Which application is the most scalable?
- Discuss the results from the previous two questions.

Rubric Score	Definition	Percent Achieving
4	**Accomplished:** Strong indication that one application implementation was more scalable than the others. Correctly identified which implementation was scalable based upon the testing results. Justification of why the implementation was scalable based upon data structures	14%
3	**Proficient:** Potential indication that one application implementation was more scalable than the others. Correctly identified which implementation was scalable based upon the testing results. Justification of why the implementation was scalable was not complete or thorough, but did hint at a data structures root cause.	50%
2	**Slightly Flawed:** Potential indication that one application implementation was more scalable than the others. Correctly identified which implementation was scalable, but not backed up by testing results. Justification of why the implementation was scalable was incomplete and did not include appropriate data structures discussion.	29%
1	**Significantly Flawed:** Incorrect identification of which application was most scalable. Did not indicate or even potentially indicate that one application was more scalable than the others. Answers indicated a lack of understanding of what scalability meant in the context of the test data.	7%
0	No response	0%

The last analysis involved determining whether or not students correctly understood which aspects of the web application impacted performance and how changing the tests would change the performance results. This grouping involved looking at students' answers to questions 14, 15, 16, 17, 24. These results are shown in Table 6.

Table 6: Analysis of questions 14, 15, 16, 17, and 24.

- Does editing the word list CSV file to change the words have any impact on the performance of the system?
- Discuss your answer to the previous question.
- Does changing the email, the first or the last name, change the performance of the system?
- Discuss your answer to the previous question.
- What happens as you run a list of misspelled words through the three different implementations?

Rubric Score	Definition	Percent Achieving
4	**Accomplished:** Responses clearly explain that changing the CSV file impacts performance depending on whether the words are spelled correctly or not and if spelled correctly, what letter the words start with. Response clearly indicates that the name has no impact on performance. Response clearly indicates that misspelled words degrade performance for 2 of the 3 implementations.	14%
3	**Proficient:** Responses indicated changes were present, but the analysis was not as complete as an accomplished response. Answer indicates no impact from changing email or name fields.	21%
2	**Slightly Flawed:** The answers indicate that all 3 implementations change in the same fashion as misspelled words are added. Answer indicates the student is unsure of the correct answer.	14%
1	**Significantly Flawed:** The answers indicate there is no change in performance, or an incorrect assessment of the change in performance due to misspelled words and changing of the email or name fields.	50%
0	No response	0%

This analysis was most puzzling. Most of the significantly flawed answers indicated that students believed that changing to a different email address significantly changed the results. Since the students were not forced to provide raw test results for this question, it is hard to determine whether the students truly saw a measurable performance difference or simply misinterpreted the results. However, from their review of the source code, students should have properly identified that the email address, along with the name information, had no impact on the application, as it simply was inserted into the POST operation and was not used by the servlet in any meaningful fashion. Most of the students properly identified the impacts of changing the word list to include either more correctly spelled words or more misspelled words, but in many cases, the depth of the analysis was not complete.

Conclusions

This paper provides a preliminary analysis of integrating an active learning exercise on performance testing into a second software engineering process course. The exercise was assigned to students with little prior development in lecture. Further, there was the expectation that they would be able to learn any additional material on their own.

Based on the analysis of student results, the students themselves could answer the questions with the lowest level of Bloom's understanding without issue. However, higher level questions caused problems in that their analysis was either incorrect or significantly flawed based on the

data.

This was the first year that this exercise was used in class, and it did not have any preparation provided. This is something that will most likely be changed in the future. From looking at the results, it appears that the students needed some form of an introduction to the topics to properly understand the questions provided at a higher Bloom's level. It is also believed that, upon further review, the questions that are asked of the students can be refined to give better results and hopefully result in better understanding of the topics.

As has been stated previously, performance testing is not one of the most important aspects of software verification and validation in the Software Engineering curriculum. This exercise does provide an active learning approach to teaching some of the core aspects of performance testing with the intent that students understand the importance of performance testing and the analysis of test results without the ability to conduct raw performance testing.

Bibliography

[1] J. Cleland-Huang, "Don't Fire the Architect! Where Were the Requirements?," *IEEE Software,* 2014.

[2] M. Dunn, "Toyota's Killer Firmware," *EDN Network,* 2013.

[3] M. Carvalho, J. DeMott, R. Ford and D. Wheeler, "Heartbleed 101," *IEEE Security and Privacy,* vol. 12, no. 4, 2014.

[4] K. Stokes, "Troubled EPAS electronic records system crashes at Queen Elizabeth Hospital creating major headache," *The Advertiser,* 2016.

[5] T. C. Lethbridge, "A Survey of the Relevance of Computer Science and Software Engineering Education," in *Proceedings 11th Conference on Software Engineering Education*, Atlanta, 1998.

[6] S. Acharya, P. Manohar, W. Schilling and P. Wu, "Enhancing Verification and Validation Education using Active," in *ASEE Annual Conference*, New Orleans, 2016.

[7] Joint Task Force on Computing Curricula, "Software Engineering 2014 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering," IEEE Computer Society, 2015.

[8] M. Towhidnejad, T. Hilburn and S. Salamah, "Reporting on the Use of a Software Development Case Study in Computing Curricula," in *ASEE Annual Conference & Exposition*, Vancouver, BC, 2011.

[9] S. Acharya, P. Manohar, W. Schilling and P. Wu, "Enhancing Verification and Validation Education Using Active Learning Tools Developed through an Academia-Industry Partnership," in *ASEE Annual Conference*, New Orleans, 2016.