

Diversity of Interaction in a Quality Assurance Course

Mark Ardis¹ and Cheryl Dugas²

Abstract - All software engineering courses face a daunting task: how to recreate within the classroom the environment of software engineering as it is practiced. There are three major difficulties to overcome: providing the cultural environment of professional software engineering, providing opportunities for learning by observation and imitation, and providing opportunities for constructive feedback from teammates. Each of these difficulties can be addressed, but some creativity may be required to solve them within the traditional classroom setting.

Index Terms - Quality assurance, situated learning, software engineering, usability testing, Vygotsky.

INTRODUCTION

Undergraduate software engineering students often have unrealistic expectations. Unless they have had some "real" experience through internships or part-time work, they often believe that software engineering is just a bigger or more technically-challenging version of the types of projects they encountered in introductory programming courses. A list of these expectations and the corresponding realities in software engineering are shown in Table I.

TABLE I
EXPECTATIONS AND REALITIES

Expectation	Reality
<ul style="list-style-type: none"> most of the actual work will be done individually, with occasional meetings to synchronize 	<ul style="list-style-type: none"> almost all work is done in teams, with occasional individual tasks
<ul style="list-style-type: none"> all team members will have the same background and experience 	<ul style="list-style-type: none"> team members have diverse backgrounds and skill-sets
<ul style="list-style-type: none"> new skills can be acquired through individual practice and reading 	<ul style="list-style-type: none"> new skills need to be acquired through observation and apprenticeship

This creates several challenges for software engineering instructors. First, they must provide the proper motivation to change the expectations of their students. Second, they need to develop the right course material. Third, they need to recreate the environment of "real" software engineering within the classroom. It is this last challenge that we address in this paper.

BACKGROUND

The Bachelor of Science in Software Engineering program at Rose-Hulman Institute of Technology contains 6 core courses taken during the junior year:

- Software Requirements and Specification
- Formal Methods in Specification and Design
- Software Project Management
- Software Architecture and Design
- Software Construction and Evolution
- Software Quality Assurance

Students take 2 courses in each quarter. A three-term capstone senior project is completed during the senior year. Senior projects usually span the entire software lifecycle and are conducted for outside clients, such as non-profit organizations or software companies.

Some software engineering skills are taught during lower-level courses. For example, students work on small team projects in almost all their computer science courses. However, many software engineering topics are not covered until the junior-level courses. Thus, it is important that students master software engineering skills during their junior year, so that they can apply them during their senior projects.

The Software Quality Assurance course covers most of the Quality and Verification and Validation topics identified in the Software Engineering Education Knowledge section of the Software Engineering Volume of Computing Curricula 2001 [1]. In addition, some Process and Requirements topics are covered. Course objectives are shown in Table II.

By comparison, many schools offer a one or two-term course in software engineering that covers all the important topics. Often these courses also include a team project. Given the large number of topics to cover these courses are usually taught in traditional lecture-style format.

¹ Mark Ardis, Department of Computer Science and Software Engineering, Rose-Hulman Institute of Technology, mark.ardis@rose-hulman.edu

² Cheryl Dugas, Department of Curriculum Instruction and Media Technology, Indiana State University, cdugas@indstate.edu

TABLE II
SOFTWARE QUALITY ASSURANCE COURSE OBJECTIVES

Bloom's Level	Objective
Application	Perform formal reviews of software artifacts
Application	Create a test plan for a software system
Application	Apply different strategies for unit-level and system-level testing
Knowledge	Understand principles and strategies of integration and regression testing
Application	Use a problem tracking system
Knowledge	Understand quality attributes of software
Application	Evaluate a user interface for suitability
Application	Automate unit testing
Knowledge	Understand purposes of quality processes, methods for measuring that quality, and standards used
Knowledge	Understand and appreciate when to use different techniques for ensuring quality products

CLASSROOM LIMITATIONS

Unfortunately, the traditional classroom reinforces the unrealistic expectations of the students. It brings together students with similar backgrounds to work independently on technical tasks. Of course, it is easy to change some of these parameters. For example, students can be grouped together to work on different types of problems.

However, there are still three major difficulties to overcome:

- recreating the diverse community of people that are found on software engineering projects
- providing opportunities for students to learn by observation
- teaching students to give constructive feedback

These three challenges reinforce one another. Students need to work with team-mates of varying backgrounds, so that they can learn from their more advanced colleagues, while helping to tutor the more junior members.

In our course we have attempted to address all three of these challenges through special exercises and a term-long project.

ENRICHING THE CLASSROOM

In order to enrich the diversity of project teams we brought in clients and users who were not students in the course. In one exercise students were asked to conduct usability testing of a system that might be used throughout campus. The subjects that participated in the testing included faculty, students from other courses, and secretarial staff. In another exercise students were asked to review deliverables of senior project teams. In a third exercise students performed process assessments of senior projects.

In all these exercises the quality assurance students were exposed to diversity of background and skill-set. This was particularly revealing in the usability testing exercise. Older participants, such as faculty and secretarial staff, behaved quite differently from younger participants.

LEARNING BY OBSERVATION

Some skills are best learned through observation. Part of usability testing involves coaching and observing a test participant. This is not an easy task, as it requires sensitivity to the participant's feelings. Too little direction leads to frustration of the participant, while too much may distort the test results. Demonstrations of bad behaviors help students see these consequences. We have used role-playing effectively for this part of the instruction.

Another aspect of learning through observation occurs when students see how various elements combine within a social context. We used role-playing to teach students about the interactions of Change Control Boards with testers and developers. This helped them see how the actions of one group affected the behavior of other groups.

A third type of learning through observation occurs when students observe other students. During the usability testing unit we assigned one team to observe another team while they conducted their tests. The observing team was given a review form with a rubric to guide their observations. This helped them focus on behaviors that they needed to master.

A fourth type of learning through observation occurs when students reflect on their own performance. We videotaped the usability testing experiments conducted by students so that they could review their performance later. Students often commented that they were unaware of some of the things they were doing until they saw the tapes.

LEARNING TO GIVE CONSTRUCTIVE FEEDBACK

As most educators know, the best way to learn something is to teach it to someone else. Part of that involves giving constructive feedback to the learner. Undergraduate students are often good at finding faults, but they are not always tactful in reporting them.

In our course we included several peer review activities to teach constructive criticism. First, students performed anonymous peer reviews of test plans created by teams of their fellow students. Each student team received at least 4 reviews of their plan. In comparing the reviews the students saw lots of good and bad examples. These reviews were also graded by the instructor.

Next, students performed anonymous peer reviews of test plans created by senior project teams. This gave them a chance to master this particular type of review. It also exposed them to the quality of work that will be expected of them on their senior projects.

Other peer review activities included code inspections and usability testing observation.

THEORETICAL SUPPORT

There are several learning theories that provide support for this style of course. These theories revolve around the social nature of the learning that occurs in the course. The earliest of these is the Social Development Theory of Lev Vygotsky [2]. Vygotsky believed that students learned best when interacting with others, especially if the others were more competent in the area being explored. The support of others allowed the student to progress farther than he/she would have alone. Vygotsky is famous for his portrayal of this gap between what a student might learn alone, and what the student might learn with support, a gap that he called the Zone of Proximal Development (ZPD).

Vygotsky also encouraged the use of models and demonstration in instruction. In the Software Quality Assurance course, the demonstration of a usability testing session by role models is an application of Vygotskian principles. Another is the use of teams to complete projects. This allows the students to take advantage of each other's strengths, and to learn and develop together. A third is the use of peer evaluation, another vehicle for students to support and learn from each other.

The second theory that can be applied to the course is the Social Learning theory of Albert Bandura [3], which is related to that of Vygotsky. Bandura emphasized the importance of learning through the observation and modeling of the behavior of others. In the Quality Assurance Course, having the students observe a demonstration of a usability session by role models is an application of Bandura's theory, as is having the students observe another student group performing its usability test. Bandura described four components to observational learning: attention to the model, retention of details, reproduction of observed behavior, and motivation to carry out the behavior. Thus, the student learns how to perform a usability test by observing, assimilating, and replicating the behavior of others.

Another theory that applies to the course is Situated Cognition, or Situated Learning Theory. This theory was first described by Brown, Collins, and Duguid [4]; and also by Lave and Wenger [5]; in the late 1980s and early 1990s. Like Vygotsky, these theorists believe that learning is socially constructed. But they also emphasize that learning is a function of the context in which it is learned. They believe that students should learn in an authentic setting, by performing authentic tasks – in what they describe as a cognitive apprenticeship. Having the students observe a usability test, and then work in teams to perform an actual usability test with authentic subjects, is an application of this theory.

EXAMPLE EXERCISES

For each course unit a sequence of exercises was designed. In most cases this began with a quiz in class, followed by a homework assignment or class role-playing, and ended with a project deliverable.

The unit on conducting usability testing provides a good example of this progression. There were 3 sessions devoted to this topic.

Session 1 consisted of a lecture and three scenarios in which a usability test was demonstrated. There was a volunteer participant who followed a usability test script. The instructor acted as the monitor. The students played the role of data logger for the first usability test and completed a log sheet quiz on what they observed. The second and third scenarios showed the same usability test, but with the monitor making mistakes. The students completed a quiz sheet on these mistakes.

Session 2 consisted of four scenarios in which students conducted usability tests, practicing the roles of monitor and data logger, and grading the performance of other students who were practicing the role of monitor. Volunteer students acted as participants, following the same script but adopting one of 4 personality types, to give the students practice with different types of participants.

In Session 3 each project team met during a class session and conducted three usability tests, performing roles of monitor and data logger.

In all 3 sessions students used the same rubric, shown in Table III, for evaluating the performance of the testing monitor. This consistency helps reinforce learning across all 3 sessions. Note that rating the performance of the test monitor requires close observation of the test participant and the test monitor. Subtle cues, such as facial expressions and tone of voice, are important in judging the experience of the test participant. Note, also, that what the test monitor does *not* do is sometimes more important than what he does.

Quizzes were used to assess student knowledge for the first two sessions. For the "real" usability tests each team was observed by another team. Each test session was videotaped, and the teams reviewed these tapes immediately after their session. The instructor used his own notes, the observation team reports, the videotapes, team summary reports, and comments from testers to assess this assignment.

TABLE III
TEST MONITOR PERFORMANCE RUBRIC

Attribute	Score
Rapport	<p>5: Excellent: participant enjoyed the testing experience</p> <p>4: Good: participant at ease during testing</p> <p>3: Satisfactory: participant not distracted during testing</p> <p>2: Marginal: participant was uncomfortable</p> <p>1: Unsatisfactory: participant hated the experience</p>
Support	<p>5: Excellent: monitor anticipated problems and solved them quickly</p> <p>4: Good: monitor helped participant whenever needed</p> <p>3: Satisfactory: monitor helped participant after it was obvious</p> <p>2: Marginal: monitor tried to help sometimes</p> <p>1: Unsatisfactory: participant was left adrift</p>
Awareness	<p>5: Excellent: monitor could probably complete the participant's sentences</p> <p>4: Good: monitor was aware of all important events</p> <p>3: Satisfactory: monitor was aware of most things</p> <p>2: Marginal: monitor missed some important events</p> <p>1: Unsatisfactory: monitor was not paying attention</p>
Pace	<p>5: Excellent: test was completed quickly, but participant was not rushed</p> <p>4: Good: testing could have gone a little faster or a little slower</p> <p>3: Satisfactory: testing was completed, but it was either too fast or too slow</p> <p>2: Marginal: testing was much too fast or much too slow</p> <p>1: Unsatisfactory: testing was too fast to yield acceptable results, or so slow that it could not be completed in time</p>

UNIFYING PROJECT

We tied all of these learning experiences together with a term-long project. The project started with review of the project requirements and ended with acceptance testing of a simple system. Along the way the students participated in many different aspects of quality assurance. Table IV lists the deliverables of the project.

TABLE IV
PROJECT DELIVERABLES

Due	Deliverable or Exercise
12/5	Requirements walkthrough
12/9	Test plan
12/16	Review of test plan
12/19	First code deliverable: initial version of system (Holiday Break)
1/6	Code inspection
1/9	Unit testing
1/16	Integration and system testing
1/20	Second code deliverable: improved user interface
1/23	Usability test plan
1/27	Usability test materials
2/3	Usability testing (including observation)
2/6	Usability testing recommendations
2/10	Third code deliverable: repairs and new features
2/13	Regression testing
2/17	Acceptance testing

Students worked in teams of 4 or 5, meeting outside the classroom to do their work.

The project component helped students appreciate the role of each type of quality assurance activity on a software engineering project. It also demonstrated the sequence of activities and their consequences.

One type of activity that was not included in the project was interaction with senior projects. This were done as "overhead", just as peer review is often viewed in the workplace.

ASSESSMENT

All 17 students completed the first offering of this course successfully. In the usability testing module students performed well on all tasks except the preparation of recommendations from testing. On that assignment one team of 4 students failed to give adequate advice to developers who would make the recommended changes. One other team met expectations for this task, and the other two teams exceeded expectations.

All of the test monitors met or exceeded expectations in performing their role during usability testing. All received high scores for rapport, support, and awareness. One test monitor was criticized by 2 of the 4 observers for taking too long in administering the test, but the instructor disagreed.

All of the data loggers met or exceeded expectations in their tasks. In some cases the logs contained many details of test subject attitudes during the tests. For example, there were notations about the facial expressions of testers in addition to the comments they made as they "thought out loud."

We believe this lesson unit reflects the education theories in the following ways:

- The students observed several demonstrations of usability tests, in which the instructors or students used role-playing. These demonstrations were held in an authentic setting. It was the same setting that was to be used when the students conducted their own usability tests.
- The students practiced performing the roles of monitor and data logger. They then planned and conducted their own usability tests in small groups, using role-playing.
- In addition, the students were recreating what they had learned/observed from the demonstration. That is, the demonstrators were acting as role models.

Session T1A

Test participants commented favorably about all of the teams. All felt that they were treated with respect, and that they felt comfortable completing their tests. Some participants offered suggestions for improving the products that they tested.

Quiz scores confirm the students' mastery of this material. Cumulative grades for quizzes on this module had a mean of 91/100. The lowest score was 71 and the highest was 99. On one quiz there were several students who misunderstood the need to rank problems by both criticality and probability. There were no other quiz questions that were answered incorrectly by more than 2 students.

LESSONS LEARNED

Students enjoyed this course and seemed to appreciate the relevance of the material to their future careers in software engineering. Comments from course evaluations include:

- "I enjoyed this course very much. I found it very easy to learn in here, and feel that the material covered in this class will be extremely useful to me in the future."
- "This is a good course for software developers and for managers. I think this course enhanced my ability to be a part of a team as well as lead a team in software development."
- "This is a great course to have... With a class like this, where its relevance is obvious, it is easy to pay attention and learn something."

The first offering of this course was a bit too ambitious. The three code deliverables added too much to the student workload. In future offerings the code will be given to the students. This has the added benefit of requiring students to test code that they have not written.

While most of the exercises were successful, a few students missed some of the finer points of usability testing. At the end of the usability test, one of the students was to do

an exit interview with the participant and have the participant complete an exit survey. Most of the students handled that well, but at least two of the students didn't. In one of the usability tests, the interviewer listened to the participant's input but didn't respond – it gave the impression that he didn't care about what the participant said – and then shoved the survey at the participant, saying "here, fill this out". In the second case, the interviewer just administered the survey without an exit interview.

In future versions of the course there will be demonstrations added in the first session showing how to conduct an exit interview and survey, and the students will practice the role of exit interviewer.

ACKNOWLEDGMENTS

We thank our colleagues in the Computer Science and Software Engineering department at Rose-Hulman for their help in the development of the quality assurance course. We are especially grateful to the participants in our usability testing exercises: Don Bagert, Tori Bowman, Merry Chambers, Amanda Chenery, Archana Chidanandan, Sue Dayhuff, Lynn Degler, Cary Laxer, Larry Merkle, Chris Moore, Arin Sarros, and Mary Wade.

REFERENCES

- [1] Sobel, A. (editor), "Software Engineering Education Knowledge", *Software Engineering Volume of Computing Curricula 2001*, ACM and IEEE-CS, 2005.
- [2] <http://tip.psychology.org/vygotsky.html>, retrieved March 19, 2005.
- [3] Bandura, A., *Self-efficacy: The exercise of control*, W.H. Freeman, 1977.
- [4] Brown, J., Collins, A., & Duguid, P., "Situated Cognition and the Culture of Learning", *Educational Researcher*, January-February, 1989.
- [5] Lave, J. and Wenger, E., *Situated Learning: Legitimate Peripheral Participation*, Cambridge University Press, 1990.