An Experience Report on Cross-Semester Student Critique and Action in an Integrated Software Engineering, Service Learning Course

Richard Burns, Terry Harvey, Lori Pollock
Computer and Information Sciences
University of Delaware
Newark, DE USA
burns, harvey, pollock @ cis.udel.edu

***Abstract*—This paper reports our experience developing a product for a real-world client using a software engineering process across multiple semesters with different students each semester. New student teams test, debug, deploy, and critique previous semesters' software and then continue its development. Students are motivated to think critically about and experience real-world software engineering practice. We describe how students in one semester collectively identified the software engineering problems that could be resolved in the current semester, how the students proceeded to tackle those problems, and the impacts of their actions for future semesters.**

***Keywords*-student critique, code reuse, service learning, whole-class software improvement**

I. Introduction

This paper describes how we integrated software critique, reflection, and code reuse into a service learning project that spans multiple semesters of classes with different students. We describe how we begin each semester with students collectively identifying the software engineering problems from the previous semesters that could be resolved for future semesters, how students proceed to tackle those problems, and the impacts of their actions. Students are motivated to think critically and experience real-world software engineering practice. We begin with a brief history.

A. History

Our project began as a brief independent study session for a small number of students with the goal of exploring the potential of the XO laptop [1] as a vehicle for teaching both software engineering and soft skills to undergraduate CS students. The students' assignment was to develop an educational learning game on the XO, a platform that was unfamiliar to them. The path to that goal was student-driven: students determined what concepts they had to master to develop such a game, distributed discovery-learning tasks across teams, and team-taught the material to other students. Not only were both qualitative and quantitative results very positive [2], but the students also reported learning at a comparable rate to other CS courses as well as maintaining or acquiring enthusiasm and confidence.

Following that pilot, we partnered with a local, 2400-student, K-8 school, targeting grades 6-8, in an underserved community that had received a donation of XO laptops. The goal of the partnership is to assist teachers at the school with integrating computational thinking into the classroom. Since Fall 2009, we have instructed and supported a full 14-week course over the fall and spring semesters with similar pedagogies to that of the independent study. There are typically 15-20 students in the course, mostly undergraduate sophomores and juniors with some basic software engineering (SE) experience. Five iterations of the class have run to date. Our goals for using service learning with undergraduates each semester are:

1) learn fundamentals of software engineering via a service context
2) increase overall learning, enthusiasm, confidence in technical abilities and communication skills
3) deliver a final educational software product that incorporates learning components and is useful to the partner school.

We describe the details of the course pedagogy, logistics, and evaluation in a recent paper [3].

There are several interesting software engineering and educational factors in the course. From an SE perspective, students experience a real client and gain from an iterative process of software design, implementation, testing, and deployment [4], [5], [6]. Student teams, paired with client teachers and peer leaders who have enrolled in the class previously [7], develop for an unfamiliar platform (XO laptop), design with new paradigms (GUI, event-driven programming), and program in a new language and framework (Python, PyGame).

Factors such as "replayability", robustness, usability, user-friendliness, and fun are essential to the client (partner school teachers and their students). During our first semester, many of our students worked very hard on developing their final game only to experience that the middle school students appreciated their game but found it boring after only a short period of time. Common complaints were amateur graphics, and uncreative and stale gameplay. Most of our initial games

featured multiple choice questions in a quiz environment.

As we reflected on the class at the end of each semester, we observed a pattern: students *had* increased their self confidence and pride in their technical skills by quickly building a product for real clients and then observing the students and teachers enjoying (somewhat) the new educational software. However, they were frustrated that the semester had ended and their game was not to the level they had wanted.

B. Outline

The remainder of this paper continues with how we introduced a two-pronged approach in the course that has been successful in motivating students to not only develop quality learning games but also to "push the envelope" and develop games which advance our project's accomplishments from previous semesters. We discuss how one iteration of our class led the initiative to develop common features that could be utilized by all games in Section III. In Section IV, we describe how we teach the API of the common framework that was developed in previous semesters to the current students. We conclude with our observations, key experiences and lessons learned from this approach.

II. Game Deployment, Critique and Student-proposed Change

A. Two-pronged Approach

We begin each semester with a two-pronged approach to engage students in the iterative improvement cycle of the software developed in previous semesters of this course. Students first analyze the software themselves and later solicit reactions from real target users.

First, students are assigned to play, test, and critique the previous semester's games. Each student develops a list of likes and dislikes from a user perspective, along with bugs they expose in their game play. As a class, we collect a list of possible improvements, making generalizations across games where possible. Teams are assigned to make some improvements to a game.

Student teams then travel to the partner school (after we first discuss soft skills relevant to interacting with school children) to observe classrooms of middle school students playing the developed learning games from the previous semester. They watch the students use the games, listen to the students' conversations with their peers, and interact with the students by asking them questions about what they like and dislike about the games. At the same time, they also obtain feedback from the teachers and note their perspective on the student game play, as they are the learning experts. The UD students write about their experiences at the partner school in a reflective journal, and we discuss their observations and reactions as a class. We collect and collate what the middle school students were doing, not doing, and saying. For most of our students, this is the first time they have systematically attempted to elicit someone's opinions and experiences.

After playing the games themselves with a critical eye and observing the middle school students playing the games, the instructor and peer leaders (students from previous semesters), provide a brief description of the architecture of each game and the process of how those games were developed. With the background of software process, architecture, and user perspective of game play, the students participate in an analysis of quality and addressing some of the concerns of the users. This sets the environment for overcoming obstacles to "push the envelope" in the current semester.

B. Case Study

During one iteration of the course, after our students both played the games from the previous semester and also observed them at the partner school, our students were tasked with modifying a game called *Cannon Fodder* to handle bugs and improve the interface. The class collectively observed that *Cannon Fodder* had a login screen with bugs, breakable menus, and incorrect behaviors for correct user responses. The class also had the following suggestions for improving the user experience: that adding a "speed control" would improve "replayability", making the answer to a question visible would benefit students who answered a question incorrectly, and adding a scoring bar would make the game more dynamic.

As an exercise, we assigned pairs of classmates a specific bug to fix in the game and allotted only one week for the fixes. Some fixes were more difficult than others. We wanted to observe the overall difficulty of this process and whether any strategies could be generalized. We also wanted to emphasize to the class that *their* software was going to be analyzed by future iterations of the class and that a good design for their source code would be essential.

During the period that our students were modifying *Cannon Fodder* with their fixes, the students reported epiphanies regarding the importance of a good overall design and code documentation. One student wrote in his reflective journal,

> "It made me think about how, [in game design], there is so much more to it than I previously thought. Slightly intimidating."

The students also naturally began working with each other across pairs as the class had to understand collectively how *Cannon Fodder* was implemented and how it could be improved. We spent class time letting students question, explain, and argue about the code. The big takeaway was that our students realized the need for a common framework. Many of their observations about *Cannon Fodder* were generalizable to other games from the previous semester. This was the first iteration of the class in which observation of a need for a common framework occurred.

Once the class recognized that many previous projects shared the same shortcomings, it was easy to steer them towards the idea of writing some code libraries to facilitate:

1) unification of the architecture of previous games
2) reduce code volume (and corresponding bugginess)
3) future development

This specific class chose to develop uniform game interfaces to: a student profile database[1] for storing preferences and game results; a leaderboard for games where teachers wanted one; a math fractions class to facilitate problem generation and displaying fractions nicely[2]; and a simple network interface to the database.

C. Reflection

We feel that the exercise of having the students interact with a previous semester's games early in the course has two important software engineering effects: that it

1) enables them to better understand the client (middle school teachers and students) and the domain (educational learning games)
2) teaches them about the importance of good software engineering such as design, modularization, code reuse, documentation, etc.

Student critique of the previous semester's games is very important. Students initially are hesitant to deliver critique knowing that in only a semester's time, their own game is going to be exposed to the same scrutiny. This has the effect of buffering criticism. However, upon visiting the partner school and explaining to the middle school students that they are going to be designing educational learning games for them, they recognize the need for high quality, and thus the need for scrutiny as well.

Students also observe that different middle school students prefer different styles of games. For example, one UD student wrote in their reflective journal:

> "...it seemed as though girls were more likely to choose games that weren't time based, while boys really went for speed games. I don't know if ...girls are less competitive, or what."

We also cover different learning styles in the course[3] and UD students also observe this at the partner school. Early observation at the partner school helps motivate why this discussion is necessary.

Most of our students also embrace that their clients are real, are eager to offer ideas[4] and are genuinely excited that someone is designing a game *for* them.

Overall, the experience of our students interacting at the partner school tremendously improves their willingness to critique, to submit their work to critique, to actively participate in class, and to produce a quality game with a well-planned game design and software architecture.

[1]Fully implemented and tested with our server but on hold until the school can host it, for privacy reasons.

[2]Middle school teachers continually ask for fraction games.

[3]We bring in education specialists to address this topic.

[4]Boys from the middle school frequently suggest basketball games while girls request shopping games.

III. Taking Action as a Class

During one iteration of the course, the class as a whole initiated the design of our first common framework that could then be extended to all games during that semester. Since the course is designed to be very active in a student-led environment, the students themselves were central in designing the game engine, brainstorming the shared parts and components, and identifying the features that would allow the current games of the semester to "push the envelope". In essence, the students motivated *themselves* to learn more advanced techniques about game design, software engineering, and the XO platform—all in a service learning setting for a real client.

A typical class had students generate specific tasks to be completed for the next stage of work in several areas. The instructor or peer leaders would lead, soliciting ideas for what had to occur next, how many people would need to work on it, and, if it was longer than one week, how to decompose it. These ideas were captured on task boards. A typical task board showed the next due dates for individual work and team work that was due for the overall common framework. Some task boards were more elaborate depending on the task. Photos of the boards were uploaded to a class wiki following each class where additional collaboration and discussion occurred.

Having students generate assignments to achieve deadlines almost completely removes complaints about assigned work. The job of the teachers becomes one of reminding students of their objectives, and occasionally walking through the scheduling consequences of decisions as a reality check. This works only because the students have bought into the semester goals for the course.

During a single week of class, a student could solve a proof-of-concept Python assignment that requires use of various game development skills (drawing, mouse use, networking, state storage, etc.), develop new game ideas based on teacher interactions (storyboarding or coding), coding part of the meta-reinforcer, and working on Python libraries. As the class progressed, it focused more on code reviews and collaborative sessions where students showed their approaches and heard suggestions from other students, peer leaders, and occasionally the instructor.

IV. Impacts on Future Student Software Engineering Experiences

The new UD class is now able to leverage the analysis and development of the previous semester by using the common framework libraries they had developed. At the point in the semester in which students are embarking on designing the architecture of their learning game, we invite students from

the previous class to present the API developed by their class to the current class. The current class is at a good point in their understanding of Python and have identified the technological hurdles for their game, and are ready to ask probing questions of the previous students to ensure they take utmost advantage of the API and avoid reinventing the wheel and hitting the same barriers as previous semesters. This is a confidence building experience for the previous student who now gets to play a teaching role.

The current students also inspect the code from the previous semesters to learn how those games were constructed to take advantage of the API. They use the previous games as maps for their games. The peer leaders also serve as a great resource for students to direct questions to when they are not sure why something was done in a particular way.

The APIs that were developed in previous semesters are continually maintained and improved based on their use and challenges perceived in their use. The original developers take bug reports and update the API. The current students also volunteer to make changes.

The identification of opportunities for new APIs by the current students through critique has had a significant impact on game quality.

An end-of-semester showcase of the learning games developed in the current semester is the capstone of the course. We and our outside evaluators have seen a significant improvement in the quality of the learning games presented at this showcase in the past two semesters. The user interfaces are of higher quality in graphics, aesthetic design, and interactivity speed. The learning components are more sophisticated as students have additional time to spend on building learning components. And, the replayability and fun factors of the games are much improved, based on our students' observations of the middle school students playing their games at the partner school.

We have observed students who were quiet, solitary coders become leaders with improved communication skills, able and willing to help others while encouraging others to participate. Current students are asking more questions in class about software reuse, architecture, and process compared to previous semesters. They are interacting more between different groups as all the groups are trying to reuse code from previous semesters.

V. Conclusion

We conclude with some key lessons learned from this approach to code critique, real-life client observation, and code reuse across semesters:

- A very powerful learning experience at the start of the course is the UD students' observation of the middle schoolers (their clients), using the software created in previous semesters. Observing game play and listening to conversations challenges UD students by giving them a deep understanding of how their game will be used and appreciated (or not). The students' search for quality is motivated by watching clients use the product, not by the instructor prodding for quality. An example:

  One former student during the observation of a middle school student playing his "finished" game at the partner school, witnessed an apparent bug crash the developed game, the middle schooler becoming frustrated and then disruptive to surrounding classmates. The misbehavior caused the middle school teacher to stop the current activity and intervene. To our former student, the takeaway was that the bug not only caused a program crash but was responsible for the propagation of the middle schooler's behavior. He later swore to never have buggy code again! The maxim of quality was understood.

- Students are stimulated to think critically about software architecture, code reuse, and software process through learning and critiquing how students in past semesters built their products, and their associated outcomes and struggles, and then brainstorming together on how to improve the overall software engineering of the products for the current semester. Students use previous students' products in intelligent ways.

Acknowledgment

We thank our evaluators, Kathleen Pusecker and Manuel Torres, for their valuable role in this project. We would also like to thank the Chester Community Charter School teachers for working with our student teams.

References

[1] OLPC, "One laptop per child," http://laptop.org/en/.

[2] L. Pollock and T. Harvey, "Combining multiple pedagogies to boost learning and enthusiasm," in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, pp. 258–262, 2011.

[3] R. Burns, L. Pollock, and T. Harvey, "Integrating hard and soft skills: Software engineers serving middle school teachers," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, pp. 209–214, 2012.

[4] J. Liu, J. Marsaglia, and D. Olson, "Teaching software engineering to make students ready for the real world," *Jouranl of Computing Sciences in Colleges*, vol. 18, pp. 43–50, 2002.

[5] J. A. Polack-Wahl, "Incorporating the client's role in a software engineering course," *30th SIGCSE Technical Symposium on Computer Science Education*, pp. 73–77, 1999.

[6] C. P. Rosience and J. A. Rosiene, "Experiences with a real software engineering client," *Frontiers in Education Conference*, 2006.

[7] B. J. Duch, S. E. Groh, and D. E. Allen, Eds., *The Power of Problem-based Learning: A Practical 'How To' for teaching Undergraduate Courses in Any Discipline*. Stylus Publications, 2001.