Open Source Contribution As An Effective Software Engineering Class Project

Robert Marmorstein
Department of Computer Science
Longwood University
Farmville, Virginia, USA
marmorsteinrm@longwood.edu

ABSTRACT

Software engineering courses often include a semester project designed to give students experience with real-world programming challenges and to expose them to phases of the software development cycle not covered in other classes. One means of engaging students in realistic programming challenges is to make participation in open source development a part of the semester project.

This paper describes an assignment in which students contribute to an open source project. The project is designed to immerse students in the open source community and expose them to the work flow and design strategies of a large project. Students work in small groups and decide both which open source community to contribute to and which specific contributions they will make. They can choose to focus on implementation of new features over software maintenance or can focus on documentation and design over both. The assignment contains a proposal phase that allows the instructor to ensure that students are exposed to a healthy cross section of the development cycle.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer Science and Information Science Education

General Terms

Design, Theory, Experimentation

Keywords

open source software, software engineering, real-world projects, team programming, software maintenance

1. INTRODUCTION

One of the goals of a software engineering course is to expose students to realistic programming challenges. It is a well-known problem[9, 1] that programming assignments in academic courses are usually very different from programming tasks in industry. While most class projects focus heavily on design and implementation, real-world projects usually require manipulation of existing code.

A common solution to this problem has been to divide students into large groups and assign them a difficult project which they complete over the course of a semester. When the instructor designs the project, this approach suffers from many of the drawbacks that programming assignments in other courses do; it is very difficult to create an assignment which can be completed in a single semester and yet is complex enough to demonstrate the application of software engineering techniques.

Furthermore, these projects typically involve implementing code "from scratch" and focus most heavily on the first stages of the waterfall model: requirements analysis, design, and implementation, rather than on testing, software maintenance, and deployment. One novel approach to this problem is described in [7], where students were introduced to the *software change process model* using code that had been developed outside the course. While this project succeeded in making the code base more realistic, the development environment was still primarily academic. It is difficult to accurately simulate the work flow, communication model, and development guidelines of a real-world project in an academic setting.

In this work, we describe a project for a one-semester software engineering course in which small groups of students interact with developers from open source projects. A unique aspect of this project is that instead of implementing a project from start to finish, students are given freedom to contribute to the project in different ways. These contributions range from implementation of new features to software maintenance tasks such as fixing bugs, writing documentation, or packaging software for distribution. Students are required to familiarize themselves with the project and submit a proposal document that ensures their contributions will expose them to a broad cross-section of the development process.

2. BACKGROUND

There have been many attempts to incorporate realism into a software engineering class project. In [8], students at geographically separate universities were paired with each other to develop visualization software for the FAA. In [6], students were organized into "virtual corporations" which simulated the management hierarchy and development process of real-world corporations. In [10], students were as-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ITICSE'11, June 27–29, Darmstadt, Germany
Copyright 2011 ACM 978-1-4503-0697-3/11/06 ...$10.00.

signed a project designed by the professor to imitate industry practices and work flow. At the University of Kentucky, students in a software engineering class developed software for tracking PKU medication for use by nurses at the medical school[3].

While most of these projects require students to produce code for real-world users, very few of them incorporate modification or maintenance of existing code. A possible explanation for this is that proprietary code is difficult to access and requires a significant time investment in the coordination between students and an industry partner.

In contrast, open source projects usually have very low barriers to entry and have well established code bases maintained by a team of experienced developers. This means that students can contribute in other ways than design and implementation of the project. Furthermore, many open source projects are sufficiently large that software engineering and project management techniques are required for their success.

Trinity University has made had success with a project that requires contributions to an open-source disaster relief application. They have incorporated open source development both into an introductory level class[5] and a software engineering course[2]. These projects were designed to engage students in service-learning and heavily emphasized the design and implementation phases of the development model, but engaged students in all phases of software production.

Faculty at North Carolina State University have had mixed success at incorporating open source into their software engineering classes[4]. In particular, they found that it is difficult to select a project that is well documented and suitable for completion in a single semester. To address some of the issues they have encountered, they established an online repository of open source projects suitable for use in the classroom. Using these projects has improved student success in their courses.

Most of these approaches require students to engage in a project selected or designed by the instructor. Since one of the most important advantages of open source is that it enables developers to "scratch their own itch", this is a serious limitation. Students are more likely to enjoy and become actively engaged in a project if they can choose something that matches their own interests. The more engaged students become in the project, the more likely they are to explore many different aspects of the design process. On the other hand, if given free reign, some students might abuse this freedom to avoid the more challenging aspects of software production. A hybrid approach, which gives students the freedom to design their own projects, but requires them to fulfill certain requirements can address both of these concerns.

3. PROJECT GOALS

Our institution's only software engineering course is a one-semester offering open to any undergraduate student who has completed the CS1 and CS2 courses. The object of the course is to introduce students to the software development processes and project management practices. The assignment of a large software project which the students work on in groups plays a large role in the accomplishment of this objective. In the past, the project has focused on the design and implementation from scratch of a web or database system specified by the instructor. In Fall 2010, we decided to approach the software engineering project in a new way.

Instead of requiring students to create software from scratch or to modify code created by the instructor, we would require students to make contributions to an existing and active open-source project. The expected advantage of this approach was that it would expose students more directly to software engineering concerns and provide them opportunities to interact and network with programmers outside the academic environment. In particular, we hoped to achieve the following four outcomes:

- Students would experience the software life-cycle from design to maintenance.
- Students would develop proficiency in using the software tools and processes used to complete a large project.
- Students would master communication skills necessary for working in a large team.
- Students would learn the importance of internal and external documentation.

We also hoped to encourage students to become a part of the open source community and build relationships with open source developers.

4. PROJECT DESIGN

Students were allowed to work singly or in pairs. Each group could earn a total of 100 points toward the final grade. Twenty of those points were earned by completing a detailed project proposal. Another twenty points came from a six to eight paragraph summary of the available documentation and development resources for their project. The remaining 60 points were earned by successfully completing a set of contributions to the open-source project.

4.1 Proposal and Resource Summary

To ensure that projects contained a wide variety of tasks, each group was required to submit a short proposal that provided background information on the community to which they planned to contribute and included a detailed timeline of planned contributions. This allowed the instructor to reject proposals which did not have enough variety and to ensure that projects maintained a reasonable level of difficulty.

After acceptance of the proposal, students were required to submit a summary of the available development resources for their project. This served two purposes. One purpose was to familiarize students with the tools they would need in order to complete the assignment and to encourage them to initiate communication with the experienced developers already working on the project. A second purpose was to encourage students to begin looking at the source code for the project as early as possible.

4.2 Open Source Contributions

In order to earn the remaining sixty points for the assignment, students received credit for the following contributions:

- Writing, testing, and submitting code for a significant new feature earned the student all 60 points if the submission was accepted by the project developers and

became part of the main source code repository. If the change was rejected by the project developers, students could still earn 30 points for a feature submission which worked correctly without crashes.

- Submitting a patch for a well-known bug earned 30 points if the patch was accepted into the source code, but only 10 points if not accepted (for coding style or other non-technical reasons).
- Reporting previously undiscovered bugs in the program and creating unit tests for known bugs earned 5 points each.
- Creating resources (such as icons, artwork, images, music, sound clips) or writing chapters of external documentation earned the student 5 points per resource.

Each group's proposal could incorporate any number of these contributions as long as the total added up to 60 points. If the initial proposal was concentrated too highly in a single area (for instance, if students wanted to only submit artwork), the professor rejected the proposal. After submitting a new, more balanced proposal, students were allowed to begin work on the resource report and the implementation of their project.

It is important to note that for most open source projects, the submission process for new features also involves working with tools for software maintenance, such as issue trackers and patch management systems. Some open source projects also require new contributions to be accompanied by unit tests. Submitting new features also exposes students to design concerns as the developers pick apart their code and uncover performance, style, and design flaws.

5. ANALYSIS AND RESULTS

Of the nine teams in the course, all but two completed the assignment with a passing grade. Five teams succeeded in getting their contributions accepted into the code base of their chosen project.

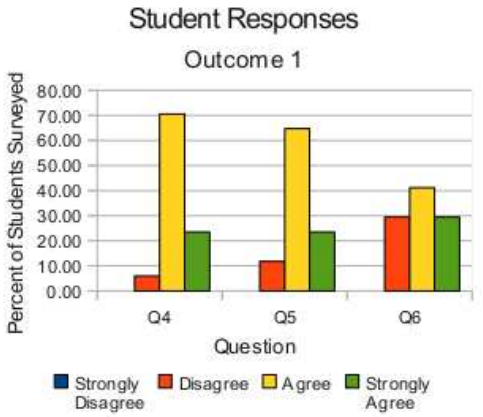
One team had particularly notable success. They completed the design and implementation of their project, a web browser plug-in for searching a music web site, very early in the semester and were able to focus on software maintenance tasks for most of the remaining time. Their software later won a best-in-class software award on a popular tech web site.

5.1 Assessment

To evaluate the success of this assignment and to assess how well we achieved the four outcomes listed previously, we performed an anonymous survey of the students. The survey consisted of sixteen statements. Students were instructed to mark 'Strongly Agree', 'Agree', 'Disagree', or 'Strongly Disagree' next to each statement. Students were also asked to approximate the total number of hours they spent working on the project.

Figure 1 shows the results of the questions used to evaluate the first outcome (students will experience the entire software life-cycle from design to maintenance).

Students almost universally perceived the project as instructive. They felt particularly strongly that they had learned about the design phase, but most of them also felt they had learned about implementation and maintenance.

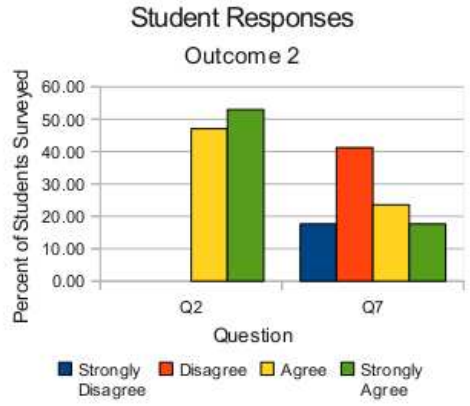


- Q4: This project taught me something about designing software
- Q5: This project taught me something about implementing software
- Q6: This project taught me something about maintaining/debugging software

Figure 1: Survey Results: Outcome 1

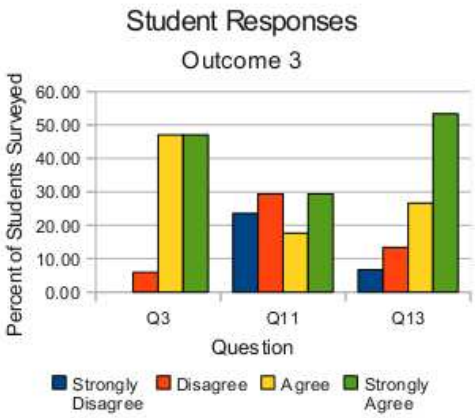
To assess the second outcome (students will develop proficiency in using software tools and processes), we used the questions in figure 2.

Every student surveyed indicated that the project had taught important lessons about software tools. However, a majority felt that they did not learn very much about working in a large team. This fact may be connected to the large number of students who indicated that the developers of the project they worked on were unhelpful (see figure 3).



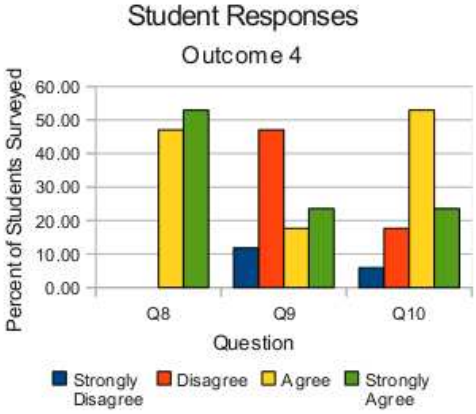
- Q2: This project taught me something about software development tools
- Q7: This project taught me something about working in a large team

Figure 2: Survey Results: Outcome 2



- Q3: This project taught me something about communicating with other developers
- Q11: The developers of the project I contributed to were helpful
- Q13: My partner and I worked well together

Figure 3: Survey Results: Outcome 3



- Q8: This project taught me something about the importance of good documentation
- Q9: The code base I contributed to was easy to understand
- Q10: The API documentation for the project I contributed to was helpful

Figure 4: Survey Results: Outcome 4

It seems likely that many of these responses were from students who switched projects halfway through the semester. Better initial project selection and screening might have significantly improved this outcome.

Figure 3 shows the questions used to evaluate outcome three (students will master communication skills necessary for working in a large team). Two students, who worked without a local partner, refrained from answering question 13.

The survey results indicate that students felt they learned a lot about communicating with other developers, but that the developers of the project they contributed to were unhelpful. They were much more comfortable communicating with their peers. A wide majority felt that they had worked well with their partner on the project.

Some of the frustration with upstream developers may be due to unclear expectations. Several teams who had patches rejected felt that the developers they communicated with did not adequately explain their reasons for rejecting the patch or provide clear instructions on how to address those issues.

On the other hand, several teams had very positive experiences with upstream developers. One team extended a solitaire game to allow the user to provide custom artwork for the card decks. After their initial submission was rejected by the project maintainer, they were able to re-factor their code into a form that was accepted as a stand-alone companion application to the project.

The questions and results for the fourth outcome (students will learn the importance of external and internal documentation) are given in figure 4. Students universally agreed that they learned that good documentation was important. The fact that many of them found the code base they worked with difficult to understand may have contributed to this understanding, especially since a significant number of students also found the API documentation for their project lacking.

In addition to questions about the four outcomes, students were asked general questions about their experience and perceptions of the project. These questions are listed in figure 5. Every student surveyed felt that they learned something from the project. A wide majority (82%) felt that they had done their best on the project, that they enjoyed the project (77%), and that the project would be helpful for finding employment (64%). However, only a few (55%) of students felt that they had started early enough on the project.

- Q1: I learned something from this project
- Q12: I did my best on this project
- Q14: I started early on this project
- Q15: The project was fun
- Q16: The project gave me experience that will help me find a job

Figure 5: Additional Survey Questions

5.2 Limitations and Future Work

One major limitation of this project is that there is very little exposure to the requirements analysis phase of software design. Students are working with established software and so their effort is heavily concentrated in the later phases of the rainfall model. Requirements analysis can be covered in other course material, but it would be nice if the project could better incorporate elements of all phases.

One way to incorporate this phase might be to require students to perform a usability study or interviews with users as part of the project proposal. The big drawback to this solution is that it extends the timeline — the students already have very limited time to complete a feature and get it ac-

cepted into the code base. Making the proposal step longer takes away valuable development time. However, this might be an appropriate improvement for a two-semester class or even for a one-semester course in which the students are organized into larger groups.

The results of the survey outline several additional limitations of the project. In particular, they highlight three major areas of concern that negatively impacted successful completion of the project: project time frame, poor communication with the open source developers, and poor documentation of the upstream project.

Many students felt that they did not have enough time to complete the project, but largely attributed that concern to not starting early enough. Groups who switched projects in the middle of the semester were particularly affected by this issue. Many of these teams finished with a passing grade, but did not get their final contributions accepted into the source code of the project. This problem strongly corroborates the experience in [4] that project selection plays a huge role in student success.

A more serious problem was that many students had difficulty communicating with upstream developers and understanding the project API documentation. Many of the teams who switched did so because of these issues. In particular, two teams of students initially attempted to work with the OpenOffice project, which had split into two competing projects (OpenOffice.org and LibreOffice). Both of these teams found communication with developers difficult, API documentation outdated, and the code base unintelligible. These teams switched to new projects, but lost several weeks of coding time.

These issues may stem partly from the students' inexperience with the technology open source projects use for communication. For example, many of the students had never used the Internet Relay Chat or subscribed to a listserv. Most large open source projects use these as their primary means of communication, blogs and social media have largely replaced these in the lives of the students. Assigning a short lab session on using these tools might be one way an instructor could facilitate better communication with the developers. Another way might be to require interviews with the project developers as part of the proposal process. The interview might include questions on project workflow and suggested coding style. The answers to the interview questions might help students be better participants in the project, but might also improve project selection in that it would "screen out" projects with a community that is unable or unwilling to take the time to interact with new developers.

The biggest drawback to these solutions is that they extend the amount of preparatory work, shorten the amount of time available for designing and implementing code, and make it harder for students to evaluate many projects. To give students more time at the end of the schedule, it might be desirable to combine the resource report with the project proposal. Merging these would eliminate a step in the preparation section of the project and get students to the implementation phase faster. Also, because the resource report requires students to take a close look at the project and initiate interaction with the project developers, this change might help students discover undesirable conditions earlier in the process.

Increasing group sizes might also make it easier for students to finish their projects on time. While a team size of two is standard in the "extreme programming" paradigm, there is no reason to limit students exclusively to one development model. Adding a third member to the group might make it easier for students to successfully complete the project on time and would mitigate another issue observed in both projects that received failing grades: one student who was not willing to pull their own weight and resulting conflict between the partners.

6. REFERENCES

[1] D. Coppit and J. M. Haddox-Schatz. Large team projects in software engineering courses. In *Proceedings of the Special Interest Group on Computer Science Edcuation*, SIGCSE '05, pages 137–141, February 2005.

[2] H. J. C. Ellis, R. A. Morelli, T. R. d. Lanerolle, and G. W. Hislop. Holistic software engineering education based on a humanitarian open source project. In *Proceedings of the Conference on Software Engineering Education & Training*, pages 327–335. IEEE Computer Society, 2007.

[3] J. H. Hayes. Energizing software engineering education through real-world projects as experimental studies. In *Proceedings of the Conference on Software Engineering Education and Training*, CSEET '02, pages 192–207. IEEE Computer Society, 2002.

[4] A. Meneely, L. Williams, and E. F. Gehringer. Rose: A repository of education-friendly open-source projects. In *Proceedings of the Conference on Innovation and Technology in Computer Science Education*, ITICSE '08, pages 7–11, 2008.

[5] R. Morelli, T. de Lanerolle, and G. W. Hislop. Foss 101: Engaging introductory students in the open source movement. In *Proceedings of the Special Interest Group for Computer Science Education*, SIGCSE '09, pages 311–315, 2009.

[6] D. Petkovic, G. Thompson, and R. Todtenhoefer. Teaching practical software engineering and global software engineering: Evaluation and comparison. In *Proceedings of the Conference on Innovation and Technology in Computer Science Education*, ITiCSE '06, pages 294–298, June 2006.

[7] M. Petrenko, D. Poshyvanyk, V. Rajlich, and J. Buchta. Teaching software evolution in open source. *Computer*, 40:25–31, November 2007.

[8] A. Rusu, A. Rusu, R. Docimo, C. Santiago, and M. Paglione. Academia-academia-industry collaborations on software engineering projects using local-remote teams. In *Proceedings of the Special Interest Group on Computer Science Education*, SIGCSE '09, pages 301–305, March 2009.

[9] M. Shaw. Software engineering education: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 371–380, June 2000.

[10] N. Tadayon. Software engineering based on the team software process with a real world project. *Journal of the Consortium for Computing Sciences in Colleges (JCSC)*, 19(4):294–298, April 2004.