

Exploiting Multiplicity to Teach Reliability and Maintainability in a Capstone Project

Janet Burge
Department of Computer Science and Systems Analysis
Miami University
Oxford, OH 45056 USA
burgeje@muohio.edu

Abstract

Many, if not most, Computer Science programs contain some form of capstone, or senior, project as a key requirement in receiving a bachelor's degree in Computer Science or Software Engineering. One decision that needs to be made is if all the students should be developing versions of the same project or if each project should be different. In this paper we describe how having multiple teams working on the same project can enrich the capstone experience by supporting reflection on the impact the quality of their code and documentation has on the reliability and maintainability of the final product.

1. Introduction

Many, if not most, Computer Science programs contain some form of capstone, or senior, project as a key requirement in receiving a bachelor's degree in Computer Science or Software Engineering. At some universities, including Miami University, capstone projects span multiple semesters and are developed in groups.

One decision that needs to be made is if all the students should be developing the same project or if each project should be different. From the instructor's perspective, there are several advantages of having all the students working on the same project. It is easier to design one project than several, it is easier to manage student teams if they are all doing the same thing, and it is easier to grade projects consistently. The question is, does this approach help the students in any way or is it just making life easier for the instructor?

Capstone courses at Miami University have used both approaches. Some instructors offer a variety of projects with each team taking a different one while others have all the students working on one project. The most recent offering of the course used the latter approach. The primary reasons for this decision were the advantages offered to the instructor listed earlier. During the course of the project we also discovered that there were many other advantages that could be exploited to provide learning opportunities for the students.

In this paper, we will describe how we were able to take advantage of multiple teams working on the same project to teach maintainability and reliability. Section 2 describes the objectives of our approach, Section 3 provides how we ran the capstone project, Section 4 presents our observed results, Section 5 describes related work and Section 6 presents our conclusions and plans for future work.

2. Objectives

At Miami University, the capstone course is considered part of the Miami Plan for Liberal Education. The Miami Plan has four key learning objectives: thinking critically;

understanding contexts; engaging with other learners; and reflecting and acting. The capstone course needs to address these objectives.

Designing and developing a software project requires *critical thinking*. There are many decisions that need to be made during development starting with the initial requirements and continuing through implementation, test, and deployment. *Understanding contexts* is important to ensure that the system meets the needs of its eventual users. Most capstone projects at Miami are developed for real customers. The project described in this paper was developed for the student chapter of a national non-profit organization. *Engaging with other learners* is necessary when developing group projects with other students. The final objective, *reflecting and acting*, is the one that is often difficult to incorporate in the classroom setting. The goal is to have the students reflect on what they have done and learned and then act on that knowledge.

Besides addressing the general Miami Plan objectives, we also wanted to try to remedy some past difficulties experienced with capstone courses: requirements surfacing at the last minute, which is often unavoidable but needs to be addressed, and student projects where all the work was performed during the last two weeks of the semester, with predictably bad results.

3. Overall design

The capstone project at Miami University is split over two semesters. During the first semester, the students take a standard software engineering course where the requirements for the capstone project are developed during the second half of the semester. The second semester then involves the design and development of the project. Some students do not continue to the second semester and take an alternative interdisciplinary capstone instead while additional students may join the class and need to be integrated into student teams.

The capstone course described here was performed by a smaller than usual class of students with only two project teams. The requirements specification developed during the first semester was a very traditional formal document containing UML use cases, class diagrams, sequence diagrams, and user interface story boards. The class was very interested in the lecture on agile methods and we discussed how that approach might be more applicable to small team projects like the type they would implement for their capstone project. Based on that discussion, and the results of a program assessment that took place at the end of the semester, I decided to try to use agile methods during the capstone implementation the following semester.

The first thing we did was to hold a Joint Application Design (JAD) Session with the customer to consolidate requirements so each team would be starting with the same set. JAD sessions are an approach used to create requirements for a new system with the involvement of all interested stakeholders [1]. There were two student teams working on two different requirements specifications that had been developed the previous semester. There were also two students new to the class who were each assigned to one of the pre-existing teams. Normally a JAD session would take more time than available in a single class period but we used the initial set of requirements and story boards to kick off the session so we were not generating requirements “from scratch.”

The student teams each summarized the JAD session in a report that gave a list of requirements that they derived from the JAD discussions. The requirements were written up on the white board and were allocated to stories [2] that would be implemented in the final system. The plan was to develop the software in four releases: two initial iterations, a maintenance release where the students swapped projects, and a

final release that consisted of any additions or repairs needed before releasing the software to the customer. The students divided the stories into three groups for the initial three releases taking into account the due dates for each release. The students were aware that the code would be swapped and when the swap would occur.

Each week the teams met with the instructor to describe their progress and demonstrate the current state of their system. Each team wrote a team status report giving their progress, what problems they had encountered, and what they expected to accomplish during the following week. Individual team members also submitted a “project diary” where they described their individual contributions to the project. The intent of this diary was to encourage them to reflect back on their progress and also to provide insight to the instructor into how the individual members contributed to their team’s progress.

For each release, the students were required to perform unit tests on the software and provide the following documentation along with their code: a test plan (due one week into the release so they could receive feedback), test procedures, on-line help pages, and a programmer’s guide. At the end of each release cycle, the students presented their project to the rest of the class and invited comments. Formal testing was performed in-class by the other team using the test procedures developed for that project. For the maintenance release, the students also wrote an individual maintenance report where they described the project, any issues they had with the installation of the other team’s code, how the software updates went, what they learned from that release compared to the previous ones, and other observations they had on the release.

The customer was invited to attend all system demonstrations. The final release consisted of selecting one project as the final deliverable and adding enhancements based on the other system and final customer requests. Either of the two projects developed by individual student teams would have been usable by the customer and both had their strengths and weaknesses. The students chose the project that required the fewest updates to the documentation. They also had a slight preference for that project because they felt the code was “better structured.” Because the class was small, all the students were able to participate in developing the final release.

4. Observed results

The first task performed by the class was the JAD session that was held in order to consolidate the requirements. Unfortunately, the only client representative who attended this session was the student organization’s faculty representative. This representative was able to give useful feedback but the session was not as productive as it could have been. At the end of the JAD session, the students and the client representative were given surveys to get their opinion on the exercise. Table 1 gives the student responses while Table 2 gives the client responses. The questions shown in this table were answered on a Likert Scale where the options were strongly disagree (SD), disagree (D), neither agree nor disagree (N), agree (A), and strongly agree (SA). In addition, all participants were asked what the most useful part of the session was and if there was anything they felt was not helpful.

The responses were mostly positive. The students found that having the customer representative present was useful and that it was helpful to go through the storyboards and get comments on them. The only comments on the question asking what was not helpful were comments about the lack of any student representatives from the client organization.

Customer involvement continued to be a problem during the semester. We were able to demonstrate one version of the initial software release to the customer at their Executive Council meeting and receive some feedback at that point. We also had one representative attend after the second release was completed. New requirements were provided by the customer at both these meetings that needed to be incorporated into the software in subsequent releases.

The software releases all went fairly smoothly. For the first two releases, the major difficulty was in getting the unit testing tool, PHPUnit2, to work (there were issues with both installation on the virtual servers and learning to use the tool). This prevented the students from performing test-first development as part of the agile methodology. The student status reports did report some use of pair programming.

Table 1. JAD session student survey

Question	SD	D	N	A	SA
I was able to express my view of the system requirements during the Joint Application Development session.				4	4
I have a good understanding of <the customer's> problem and how to solve it.	1		1	2	4
The JAD session helped me understand the problem better than before.			2	2	4
If the system developed conforms to the requirements we described today, it will be useful.			1	2	5

Table 2. JAD session customer survey

Question	SD	D	N	A	SA
I was able to express my view of the system requirements during the Joint Application Development session.					1
The presentations given by the students were effective at expressing their view of the system requirements.				1	
I feel that after the session the students have a good understanding of the problem and how to solve it.					1
If the system developed conforms to the requirements we described today, it will be useful.					1

The students enjoyed the in-class testing portion of the class, especially the first session. In that session, both teams were able to successfully perform HTML injection attacks on the other team's project. For the second session, after the second release, one of the teams had tightened security enough that these attacks were not able to succeed. While there were still some issues with handling error cases, both teams had completed all the basic functionality required for their releases and had created usable systems.

The maintenance release also went fairly well. Many of the students commented that having worked on their own version of the system was a key reason why the

maintenance was easier than they had expected. There were still some difficulties reported in their maintenance reports:

- One team had not used SVN (the configuration management system) correctly and the other team had trouble initially accessing their code.
- One team was very sparse with their comments. The other team was able to figure it out because they had a good idea of what it was supposed to be doing.
- There were some differences in the data model and naming conventions but they were easily understood.
- The teams had some difficulties understanding the other systems' structure.

Neither team made much use of the Programmer's Manual written by the other team although several students commented that if they had not already understood what the system was supposed to do they would have probably used it more.

The students were all asked to reflect on what they could have done differently in their earlier releases based on their experience with the maintenance release. The observations included:

- Since the code was the first place they went to, they probably should have written more comments.
- The installation guide needs to be aimed more at novices.
- Variable name inconsistencies were frustrating – the same concept (such as a student ID) should have the same name throughout.
- Keeping around a lot of extra code ("garbage files") made it difficult to figure out what code was really being used in the system.
- Test files and test code should be identified as such and organized.
- File naming conventions should be used so it is clearer what the function is.
- Both teams noticed some project-specific improvements that could be made based on how the other team implemented some of their functionality.

A survey was given at the end of the course to get student feedback. Table 3 gives the results for the eight Likert Scale questions. In addition to the questions shown in the table, the students were also asked if making changes in the other team's code was more difficult than expected. Five found it to be about what they expected and three found it to be easier than expected.

As shown in Table 3, allocating stories to releases, performing multiple releases over the course of the semester, performing formal tests, and maintaining the other team's software were all given positive reviews. The one portion of the class that the students did not find valuable was the unit testing. This could be because they had so much difficulty with getting the unit testing tools working. The students were responsible for maintaining their own development environments for the course. This contributed towards their learning but also made the project development more difficult.

Table 3. End of course survey

Question	SD	D	N	A	SA
Working as a group to define stories and allocate them to releases helped us to allocate our time more effectively.			1	2	5
Working as a group to define stories and allocate them to releases made us feel like we had more input into our capstone experience.			1	4	3
Having multiple releases resulted in a better software product at the end of the class.				1	7
Writing and executing unit tests resulted in better software.		4	4		
Writing and executing formal system tests resulted in better software.		1	1	2	4
Knowing that the other group would end up modifying my system made me pay more attention to system structure.			2	6	
Knowing that the other group would end up modifying my system made me pay more attention to my documentation.		1	1	5	1
Performing maintenance on the other team's code made me more aware of what I could do to make my own work more maintainable.			1	2	5

5. Relationship to Related work

There have been many papers in recent conferences describing methods for administering capstone projects. Several have stressed the importance of realistic projects [3][4]. The project here followed those recommendations by using a real project for a real, although internal, customer. Some have included agile techniques such as pair programming [5] and Test Driven Development [6]. This project encouraged students to do pair programming, although it was not enforced. As mentioned earlier, Test Driven Development was not successful because the teams had difficulty installing and using PHPUnit2.

Numerous universities have incorporated maintenance projects into their software engineering curriculum in different ways. Software maintenance courses have been used to give students maintenance experience and to allow realistic projects to be developed in the time-frame given by a single class.

One example of a maintenance-based project course was a year-long class on software maintenance where the students worked on making maintenance enhancements to existing projects, both for industrial clients and by contributing toward open source projects [7]. This project exposed students to real-world development issues like clients who changed their requirements frequently. The primary focus was on the projects themselves. The class described here had the students designing and building software as well as performing the maintenance exercise. Another maintenance-based project course used the same project over multiple semesters of the same course. The project involved updates to a Pascal interpreter that contained numerous bugs [8]. The students were required to follow rigorous software development standards. This project gave the students an idea of the size of a real project. Students also had to modify

software developed by students in earlier courses on the same system. One team was forced to update their software after the instructor pointed out that it would be difficult to maintain; another team left their successors with code that had to be thrown out completely. The project described in this paper had the maintenance taking place in the same semester as the original development.

A more lecture-oriented approach was used in a class where the students were taught maintenance oriented concepts such as metrics, configuration management, re-use, and risk analysis [9]. The students did homework assignments targeted at specific concepts in a variety of programming language as well as working on a group project updating existing code. The course managed to convert some students to the view that documentation is useful. In the course described in this paper, the students had already been taught maintenance concepts in an earlier software engineering course.

These courses did not have students practice maintenance using a different team's version of the same project in the same course. There was a study done that had students swap projects at the end of the analysis phase [10]. The students had not been warned of the swap in advance and the swap was met with some hostility. The students did decide at the end of the course that the exercise was beneficial. In this case, they were not adding on to the software but were designing and implementing a system where the requirements were defined by a different team.

6. Conclusions and future work

Reliability and maintainability are aspects often sacrificed in the rush to meet deadlines. The experience of dealing with the consequences of these sacrifices from a user perspective (when testing) and a developer perspective (when maintaining) teaches the students far more than simply getting a low project grade at the end of the semester. The students learned the impact of insufficient error checking during the in-class testing and gained a new appreciation for documentation and good system structure when maintaining each others' code.

A large factor in the success of this project, particularly the system swap, was that the students were all working on the same application. Several of the students commented that trying to modify the other team's code was much easier when they were already familiar with the problem that it was trying to solve. While this can be viewed as making the task easier and less realistic, it also gave the students a direct basis of comparison when looking back at their own work. It allowed them to experience some of the difficulties and challenges involved in software maintenance yet still end up with a working software system at the end. Similarly, problems they encountered in testing each other's software could be applied directly to their own project.

The class described in this paper was run with only two teams of students. This made the project easier on some ways by giving students time to present their work in-class and have student meetings with the instructor. On the other hand, it did add an additional element of risk where if one team failed to successfully complete their releases it would have jeopardized the in-class testing and the maintenance swap. Scaling the project to a larger class would cause some logistical difficulties for team meetings and group presentations. There would need to be additional time scheduled for those course components. With more teams, if a project was not completed in time for testing or maintenance it might be possible to not include that project in subsequent releases. There could also be changes in team personnel between releases if necessary. This would give the students the additional experience of

having to adapt to changing team structure mid-project which is something they would invariably encounter out in the real world.

The capstone class was viewed as a success by the instructor and the students. While the students did not enjoy every aspect of the course (particularly the frustrations with the unit testing tools), overall the reaction was very positive. The students felt more involved in the project by being able to help decide what the releases would consist of, they enjoyed trying to break the other team's project during formal testing, and they felt that having multiple releases made it easier to spread the work out over the semester and resulted in a better system at the end of the class. The students also had ample opportunity to "reflect and act" by looking back at their past progress and using what they learned in subsequent releases of the system. The end result was two working capstone projects that were sufficient to meet the customer's needs – a successful conclusion to the students' college experience.

7. Acknowledgements

Having the students test each others' projects is an idea that was used by Dr. Scott Campbell when he taught the capstone course and he was the one who proposed that as a key component to meeting the "reflecting and acting" objective of the Miami Plan. Some of the ideas for the capstone course described here resulted from a program assessment with colleagues Dr. Campbell, Dr. Kiper, and Dr. Schaber, all past instructors of the course. I would also like to thank my capstone students for their hard work and participation in the experimental course.

8. References

- [1] Bruegge, B. and A.H. Dutoit, Object-Oriented Software Engineering Using UML, Patterns, and Java, Prentice Hall, 2004.
- [2] Beck, K. Extreme Programming Explained, Addison-Wesley, 1999.
- [3] D.J. Frailey, "Bringing Realistic Software Engineering Assignments to the Software Engineering Classroom", Conference on Software Engineering Education & Training, Oahu, Hawaii, 2006, pp. 51-60.
- [4] R.J. Fornaro, M.R. Heil and A.L. Tharp, "What Clients Want - What Students Do: Reflections on Ten Years of Sponsored Senior Design Projects", Conference on Software Engineering Education & Training, Oahu, Hawaii, 2006, pp. 226-236.
- [5] L. Layman, "Changing Students' Perceptions: An Analysis of the Supplementary Benefits of Collaborative Software Development", Conference on Software Engineering Education & Training, Oahu, Hawaii, 2006, pp. 159-156.
- [6] D.S. Janzen and H. Saiedian, "On the Influence of Test-Driven Development on Software Design", Conference on Software Engineering Education & Training, Oahu, Hawaii, 2006, pp. 141-148.
- [7] J.H. Andrews and H.L. Lutfiyya, "Experiences with a Software Maintenance Project Course", IEEE Transactions on Education, Vol. 43, No. 4, Nov. 2000, pp. 383-388.
- [8] K.R. Pierce, "Teaching Software Engineering Principles using Maintenance-based Projects", Conference on Software Engineering Education and Training, Virginia Beach, Virginia, 1997, pp. 53-60.
- [9] J. Slimick, "An Undergraduate Course in Software Maintenance and Enhancement", Conference on Software Engineering Education and Training, Virginia Beach, Virginia, 1997, pp. 61-73.
- [10] A. Young, and S. Mann, S., "Innovation in Software Engineering Education", Conference on innovation and Technology in Computer Science Education, Aarhus, Denmark, 2002, pp. 219-219.