International Journal of Cyber Society and Education
Pages 31-38, Vol. 8, No.1, June 2015
doi: 10.7903/ijcse.1238

INCORPORATING SOFTWARE MAINTENANCE IN A SENIOR CAPSTONE PROJECT

Ira Weissberger
University of Virginia's College at Wise
1 College Avenue, Wise, VA 24293, USA
imw3h@uvawise.edu

Abrar Qureshi
University of Virginia's College at Wise
1 College Avenue, Wise, VA 24293, USA
aaq3e@mcs.uvawise.edu

Assad Chowhan
University of Virginia's College at Wise
1 College Avenue, Wise, VA 24293, USA
aac4c@uvawise.edu

Ethan Collins
University of Virginia's College at Wise
1 College Avenue, Wise, VA 24293, USA
ecc9e@uvawise.edu

Dakota Gallimore
University of Virginia's College at Wise
1 College Avenue, Wise, VA 24293, USA
dmg6t@uvawise.edu

ABSTRACT

Software engineering capstone projects give students experience in developing a software product throughout the software life cycle. Projects such as these give students practical experience in applying concepts they have learned in their software engineering and computer science classes. This paper presents a software engineering capstone project conducted by students at the University of Virginia's College at Wise. The work of the students is documented in this paper. Unlike previous capstone projects conducted at this university, however, this one uses methods commonly found in agile development and is an adaptive maintenance development effort.

Keyword: Capstone Project, Software Engineering, Agile Development, Mobile Application, Software Maintenance

INTRODUCTION

Software engineering continues to be a growing field. From 2012 to 2022, the employment of software developers is expected to grow at least 22 percent, which is much higher than the average for all occupations (Occupational Outlook Handbook, 2014-15 Edition, Software Developers, 2014). Even among careers in Science, Technology, Engineering, and Math (STEM), those in computers are expected to outnumber those of other fields. A report from the Georgetown University Center on Education and the Workforce forecasts that 51 percent of STEM occupations will be computer-related by 2018 (Carnevale, Smith, & Melton, 2011). It is useful for students who are going to become software engineers to have some practical experience before graduation.

Software engineering capstone projects introduce students to develop a software product over the course of two semesters. Students typically design a new piece of software and are responsible for everything from developing requirements to verifying that the software meets those requirements. Vanhanen, Lehtinen, and Lassenius (2012) have shown that capstone projects give students valuable experience by providing work in which they can acquire necessary software engineering skills. Broman, Sandahl, and Abu Baker (2012) stated that these skills can then be applied to jobs in industry. Other capstone courses incorporate process-oriented aspects and practices of software engineering (Fagerholm & Vihavainen, 2013). The capstone project can also be adapted to fit a maintenance project, where the students port an existing application to a different platform. These types of maintenance projects can also start by gathering requirements and following a normal software process.

This paper presents a case study done on the work performed by a team of students during the Fall 2013 and Spring 2014 semesters for their software engineering capstone project. This capstone project was conducted at the University of Virginia's College at Wise. During the Fall 2012 semester, a different group of students developed a web-based Fleet Management System (FMS) so that people could reserve a car to rent from the college. Before the students implemented this system, the college had been using a paper-based system to take reservations. Although previous capstone projects have developed products from initial concepts, this capstone project was an adaptive maintenance project. The students took an existing web application and ported it to the Android platform, so that people with mobile devices could use the same functionality. The paper focuses on the development methods the students used to complete part of the project (through the end of code construction and beginning of system testing) and could be used as a template for future software engineering capstone projects.

SOFTWARE DEVELOPMENT

Requirements Engineering

Chen, Hong, and Chen (2014) have shown that encouraging students to work in a more structured approach during software development leads to better results in the capstone course. This is similar to the way that this capstone course was conducted. For requirements development, the team followed the customary process of elicitation, analysis, specification, and validation. After creating a baseline set of initial requirements, the team then moved into the requirements management phase where each change request was considered but not implemented immediately. Techniques such as interviewing, ethnography, and developing use cases are all good methods to discover requirements for the software system to be constructed. An important requirements discovery technique for a replacement or a port of an existing system, however, is to interact with the existing system to observe how it works and which requirements will need to be retained in the new system.

The existing FMS was the senior project during the 2012 school year at the college, and it failed to reach deployment. Therefore, the team was tasked with deploying the website. Once the website went live, the team could interact with its functionality and data as a way of eliciting requirements. However, the team also decided to also rely on other means of requirements elicitation, such as interviewing the client. As is usual with agile methods, the application is developed in very small increments, with a new version of software appearing approximately every two to three weeks. The team used this strategy to implement the requirements which were gathered and then scheduled more meetings with the client to elicit additional requirements.

During the next phase of requirements development, the team analyzed the requirements that had been elicited. From the requirements that had been gathered, the team created an initial set of use cases. During the analysis of the requirements received from the client, the team found that some of the requirements needed clarification. The team had to backtrack to the requirements elicitation phase to refine those requirements. Such backtracking is to be expected in requirements development. Rarely do all four phases of requirements development (elicitation, analysis, specification, and validation) occur in a one-time linear sequence. In most cases, several iterations must occur before arriving at a stable set of requirements which can be used for design and implementation. After the analysis phase was complete, the students began working on the next phase: developing the Software Requirements Specification (SRS) document. This document was the product of the requirements elicitation and analysis phases. The project scope had then been set, and the requirements were ready for validation by the client.

Code Construction

The use of agile development techniques meant that the team skipped the majority of software design. Since agile principles state that design is not as important since the code will be going through rapid changes and refactoring, no detailed design phase was conducted in this capstone project; this is another distinction between this process and previous projects. The team aimed to put out a new version of software to the customer approximately every three weeks as is customary using agile methods. However, as Mahnic (2012) has shown, traditional plan-driven approaches to software engineering can be combined with more agile ones. Thus, in addition to creating a formal requirements specification and performing requirements traceability, the students suspended the release of a version of software in order to produce design documentation. The advantage of this kind of reverse engineering has the benefit that if the original developers are no longer around to support the software, the new developers have an easier time understanding the software by looking at the design documentation rather than by viewing the source code.

The students used this approach to provide design documentation once they had verified most of the requirements for a specific release of the software.

Code reviews were also done informally after completion of every large piece of functionality. Code reviews helped the team clean up their code by making it more efficient and more amenable to future updates or maintenance. The code reviews proved to be a good way to ensure that the team produced a quality product and had met the requirements that the client had validated. This was all in line with the agile principles of the developers continuously improving the code to provide for easier maintenance in the future.

Software Testing

Most of testing done in the first semester and at the beginning of the second was unit testing. As of this writing, the team is constructing a system test document which will be used to verify all the requirements at the end of the second semester. They have been developing system test plans based off of the use cases they have created in the requirements engineering phase of development. In addition to the development team running system tests at the end of the second semester, two independent testing teams from another class will construct their own system test plans on the basis of the requirements. These independent test teams will then run their tests against the final version of the software developed. Independent test teams have several advantages. One advantage is that an independent test team may think of ways to test the software that the original development team may have overlooked. Another advantage is that there is no conflict of interest from independent test teams since they did not develop the software. Sometimes developers are reluctant to show that their software does not meet the requirement specification so they may not test it as exhaustively as they should. In the system test document, the students must create a requirements traceability matrix to make sure that every requirement has been mapped to at least one test case and therefore, every requirement should then be verified.

PROJECT MANAGEMENT

Project management is concerned with controlling resources and risk to create a software product which is on time, under budget, and meets the needs of the customer. For this capstone class, the students focused on risk management techniques. Initially, the team was given a template for the Risk Mitigation Plan. With this template, they were able to see how they were expected to evaluate the risks for the project. Other than providing the introduction of the Risk Mitigation Plan, the team was expected to identify each risk, list the terms for monitoring that risk, and elaborate on the mitigation of all risks that could arise.

First, the team researched risk evaluation techniques. They found some metrics that fit the project and were proper for the risks identified. The team found a way to score each risk, and based upon the score, calculated the impact and probability rating of the risk occurring during the project (Table 1). The impact of each risk was evaluated first, based upon a scale of values from one to ten (one being negligible impact and ten being critical impact). Next, the team evaluated the probability of the risk occurring based upon a scale from one to five (one being very unlikely and five being very likely).

Table 1 Probability and Impact Risk Rating Matrix

	Impact				
Probability	Negligible (1)	Minor (3)	Moderate (5)	Serious (8)	Critical (10)
Very Likely to Occur (5)	5	15	25	40	50
Probably will occur (4)	4	12	20	32	40
50% chance of occurring (3)	3	9	15	24	30
Unlikely (2)	2	6	10	16	20
Very unlikely to occur (1)	1	3	5	8	10

Finally, based upon the two values assigned to each risk, the team calculated a “Risk Score” based upon the product of the impact and the probability. With each score for each risk, they were able to place the results into a table with all the scores highlighted. The risks, identified along with their probability, impact, and total risk score are shown in Table 2.

Table 2 Identified Risks and Risk Scores

Impact Description	Impact	Probability	Numeric score
Working around classmate schedules	Moderate (5)	Very Likely(5)	25
Working concurrently on other projects	Serious(8)	Very Likely(5)	40
Time constraint including college breaks and holidays	Moderate(5)	Very Likely(5)	25
Scheduling meetings that work with client schedules	Moderate(5)	Probably will occur(4)	20
Learning how to develop for Android Mobile	Critical(10)	Very Likely(5)	50
Android Fragmentation	Critical(10)	Very Likely(5)	50
Possibility of potential hosting issues depending on IT Department decisions and limitations	Serious(8)	Could Occur(3)	24

CONCLUSIONS

This paper presents the first half of a senior capstone project on gathering requirements and developing initial versions of the software using agile principles of development. The team has successfully applied the principles they have learned in previous software engineering classes. Similar to previous capstone projects conducted at this university, having a software process for various phases of development helped the project to succeed. All future capstone projects would benefit from a standard software process describing the activities, inputs, and outputs from every phase of the life cycle regardless of what development technique was used (e.g. waterfall, spiral, agile). The software process could then be tailored, first to a specific development technique and then to each capstone project. This is how the software process is used in industry. Another way to benefit capstone projects is to have students and professors fill out a lessons learned document at the conclusion of each capstone project. Identifying the project's successes and failures could improve the capstone process by incorporating series of corrective and preventive actions. This continuous improvement of the software process would result in richer learning experiences for students working on future capstone projects.

REFERENCES

Broman, D., Sandahl, K., & Abu Baker, M. (2012). The company approach to software engineering project courses. *IEEE Transactions on Education, 55*(4), 445-452.

http://dx.doi.org/10.1109/TE.2012.2187208.

Carnevale A.P., Smith, N., & Melton, M. (2014, February 12). *STEM – Science, technology, engineering, mathematics.* Retrieved from http://cew.georgetown.edu/STEM/.

Chen, C.Y., Hong, Y.C., & Chen, P.C. (2014). Effects of the meetings-flow approach on quality teamwork in the training of software capstone projects. *IEEE Transactions on Education, 57*(3), 201-208. http://dx.doi.org/10.1109/TE.2014.2305918.

Fagerholm, F., & Vihavainen, A. (2013). Peer Assessment in experiential learning. In C. Dyer (Ed.), *Proceedings of the 2013 IEEE Frontiers in Education Conference* (pp. 1723-1729). Oklahoma City, Oklahoma: IEEE Press. http://dx.doi.org/10.1109/FIE.2013.6685132.

Mahnic, V. (2012). A capstone course on agile software development using Scrum. *IEEE Transactions on Education, 55*(1), 99-106. http://dx.doi.org/10.1109/TE.2011.2142311.

Occupational Outlook Handbook, 2014-15 Edition, Software Developers. (2014, February 12). Home: Occupational Outlook Handbook: U.S. Bureau of Labor Statistics. Retrieved from http://www.bls.gov/ooh/computerand-information-technology/software-developers.htm.

Vanhanen, J., Lehtinen, T.O., & Lassenius, C. (2012). Teaching real-world software engineering through a capstone project course with industrial customers. In W. Weck and N. Seyff (Eds.), *Proceedings of the 2012 First International Workshop on Software Engineering Education based on Real-World Experiences* (EduRex) (pp. 29-32). Zurich, Switzerland: IEEE Press. http://dx.doi.org/10.1109/EduRex.2012.6225702.