The Software Engineering Capstone: Structure and Tradeoffs

A.T. Chamillard
Department of Computer Science
U.S. Air Force Academy, CO 80840
703-428-0528
achamillard@hq.dcma.mil

Kim A. Braun
Department of Computer Science
U.S. Air Force Academy, CO 80840
719-333-3590
ksbraunco@bigfoot.com

Abstract

One of the largest challenges facing educators teaching undergraduate software engineering courses is providing the students with meaningful experiences they will find useful when they complete their undergraduate education. Those experiences should include all phases of the software development process, and to be as realistic as possible they should also include the uncertainty and continual change present in any real project. In addition, those experiences need to include working with others in a team, which can affect the morale of some students and also poses challenges to the professor when the time to assign grades arrives. In this paper we discuss how we have tailored the software engineering capstone sequence at the U.S. Air Force Academy to address these issues.

Keywords

Software engineering, software development, individual evaluation, group work, student management teams.

1 Introduction

All students majoring in computer science at the U.S. Air Force Academy (called USAFA hereafter) are required to take a two-semester capstone sequence in software engineering -- Comp Sci 453 and Comp Sci 454. While we have incorporated some software engineering concepts throughout the computer science curriculum, the 453/454 sequence provides the majority of student experience with software engineering issues.

Although we believe it is important to ensure our students are familiar with the technical issues associated with software engineering (e.g., requirements analysis, design, testing, formal methods, and software metrics), we are convinced that it is just as important to expose them to the management issues associated with software development.

SIGCSE '02, February 27- March 3, 2002, Covington, Kentucky, USA.
ACM 1-58113-473-8/02/0002.

Because graduates from USAFA often enter project management positions immediately upon graduation, it is critical that our students also understand the management challenges they will face as project managers. We therefore cover and require the students to use numerous management concepts in the sequence; these include general project management, configuration management, and quality assurance concepts, in addition to more detailed management tasks like cost estimation and project scheduling. We view a balance between technical and management issues as a high priority, so our capstone sequence addresses both issues.

We also want our graduates to have some experience dealing with real problems rather than simple, or even complicated, academic exercises. Because some academic departments at USAFA provide support to active military organizations, we have been able to assign our students real problems to solve. Using real projects helps students accept the importance of developing a working solution to the selected problem and also gives them experience dealing with a real customer.

The next section describes the projects we have included in the sequence over the last three years, and Section 3 presents details about our course structure and the required student activities. Section 4 discusses numerous tradeoffs we have made in the sequence in terms of the level of guidance we provide to the students, our techniques for evaluating individual contributions in the group project, and the implications of other aspects of our approach. The final section presents our conclusions.

2 Project Descriptions

The most critical aspect of the 453/454 sequence is the use of real projects, with real customers. Others have already pointed out the benefits of using such projects in software engineering courses [1, 7-10], so in this section we simply describe the projects we have used in the past several years. While some of our customers turn out to be other academic departments at USAFA, the projects are always developed to support real users accomplishing real tasks.

The project for the 1998 Academic Year (AY) was the development of the ground station software required to fly a satellite (called FalconSat) being launched by the Department of Astronautics. The FalconSat Ground Station

(FGS) needed to provide commanding and telemetry processing capability for the satellite, as well as providing a Graphical User Interface for the ground station. All commands were sent and all telemetry was received on a serial port. The FGS software was hosted on a standard PC. Students interacted with representatives from the Department of Astronautics to gather the requirements for the software.

In the 1999 Academic Year the Aeronautics department was developing an Unmanned Aerial Vehicle (UAV) in conjunction with a senior-level engineering course at USAFA. They contracted to purchase a ground control station for the UAV, but to fully test that ground station, and to help them with development of the UAV software, they needed a UAV Simulator. The UAV Simulator (UAVS) needed to provide the processing capability to respond to commands (both joystick and autonomous operation), to update UAV state (position, velocity, etc.) appropriately, and to transmit telemetry back to the ground station. All commands were received and all telemetry was sent on a serial port. The UAVS software was hosted on a standard PC. For this project, students interacted with a representative from the Department of Astronautics to gather detailed requirements for the simulator.

The project for the 2000 Academic Year was the development of a Computer Enhanced Air Traffic Control Management System (CEATMS). This was a touchscreen system developed to track the location of aircraft taxiing and flying at USAFA. The system involved multiple screens and multiple users, and was designed to replace the strips of paper commonly used in most towers at small airfields. Students worked with air traffic controllers at the USAFA airfield to gather the requirements for the system.

3 Course Structure

In this section, we discuss some of the important structural decisions we have made for the course. Although the 453/454 sequence actually consists of two courses, we structure the sequence and approach the material as though the sequence is a single, integrated two-semester course. In this and the following section, our references to "the course" therefore are actually references to the 453/454 sequence. While there are of course tradeoffs associated with our structural decisions, we defer discussing the major course tradeoffs until the following section.

One key component of the course structure, especially from the students' perspective, is the distribution of graded work between individual work and group work. The development of the systems required in the course is inherently a group activity, but it is difficult to assess each individual's contribution to the product generated by the group (this problem is discussed further in the following section). As instructors, we would like to ensure that the course grade for each individual represents an evaluation of both their individual work and their contribution to the group project. We have therefore distributed the grading percentages in the course evenly: 50% of the grade is based on individual work and 50% of the grade is based on group work. The individual work is comprised of quizzes, tests, and exercises, while the group work is comprised of the project activities associated with developing the required system.

We also need to address the organizational structure for the project; how large are the teams, and how are they composed? The number of students in the course (for the three years addressed in this paper) has ranged from 20 to 31 students in one or two sections. To build a team for the project, we break the students in a particular section into smaller groups, where each group is responsible for the development of a particular subsystem for the project. Based on enrollment in the section and the size of the subsystem, these groups range in size from two to four students. Separate from the subsystem groups, we also assign several "management staff" for the section: a project manager, a system engineer, a configuration and quality assurance manager, and an independent system tester.

Given the organizational structure described above, it falls on the instructor to determine how to place the student into the management positions and the subsystem groups. We have approached this problem in two different ways, both of which seemed to be effective.

In the 1998 and 1999 Academic Years, we asked each of the students in a section to provide a list of their top three choices for the management positions and the subsystems on which they'd like to work. We then made the assignments into the positions and groups based on student preferences. We note, however, that we used some anecdotal information when placing students into key positions, particular as project manager and system engineer. To fill these positions, we discussed the capabilities of the volunteers for those positions with instructors who had taught them in previous courses to assess whether or not we could reasonably expect them to effectively fulfill their responsibilities in those key positions.

In the 2000 Academic Year we had a single section that we separated into two teams. As in prior semesters, we selected the management staff for each team based on their preferences and discussions with other instructors in the department. We also interviewed the project manager for each team before finalizing their assignment to that position to ensure they understood the effort involved in that position. After we selected the management staff for each team, the project manager placed the rest of the team members into the subsystem groups (though we did approve the organizational structure each project manager developed).

One of the most unique characteristics of our course is our use of a Student Management Team (SMT) as a course feedback and improvement mechanism. We began using SMTs in the 2000 Academic Year. The SMT consists of four students from the class: a lead, a recorder, and two members. The purpose of the SMT is to consolidate student feedback about the course and to meet regularly with the instructor in an ongoing process of course review and improvement. The instructor also treated the SMT as a resource to help solve real or perceived problems in the course (like the individual contribution evaluation discussed in the following section). The students benefit from the SMT because the course can be improved as a result of their efforts. Additionally, students feel more like active participants in their learning when they can actively work with their classmates and the instructor to implement positive change in the course. The instructor benefits both as a result of improvements in the course and from improved student motivation. Because the SMT met once a week and met with the instructor every other week, the SMT did not represent a large time drain for either students or the instructor, and the benefits of using the SMT were well worth the time required.

The use of SMTs needs to be approached with some care, however. The instructor needs to be open and candid about why particular SMT suggestions will or will not be implemented, and also needs to be appropriately firm so that students do not believe they have "taken over" the course. It is also important that the SMT solicit inputs from the other students in the course; otherwise, the SMT will be viewed as an elite group of teacher favorites, and much of the potential benefit of the SMT will be lost.

Another unique course feedback technique we incorporated was our use of a focus group, consisting of all the students in the course. In AY 2000 our course was the first course in the Basic Sciences division at our school to use a focus group. The instructor met with members of our Center for Educational Excellence (CEE) prior to the focus group meeting to develop a set of questions for the group. These questions were designed to give the instructor meaningful feedback about the course. The CEE members then met with the focus group to discuss those questions without the instructor present. Because the CEE then provided a written report containing anonymous comments for the instructor, the focus group gave the students an opportunity to provide anonymous, meaningful feedback to the instructor without being concerned about retribution.

4 Course Tradeoffs

There are numerous tradeoffs to be made when developing any software engineering course, or any course in general. In this section, we discuss some of the tradeoffs we have identified and the approach we use to address them. While we believe we have identified the tradeoffs with the most potential impact on the course, there may be other issues that we have overlooked.

One of our tradeoffs addresses the level of guidance we give to the students in the course. We believe there is extensive value in letting the students experience both the successes and failures associated with a real development project, and in many cases we do not provide as much guidance as the students would like. Students become upset when they discover they made a mistake that leads to rework later on, and they typically believe that the instructor should have prevented them from making any such mistakes. To address these issues, we tend to provide more guidance early in the project then leave the students "on their own" more in the latter part of the course. With this approach, we can work on developing good student understanding of the concepts early in the course, but also gain the benefits of having the students make and learn from real mistakes.

A related tradeoff addresses documentation format and content. Based on the probability that our students will be involved in government projects after graduation, we have them develop their documentation using standardized formats (Mil-Std 498 DIDs for the 1998 and 1999 AYs and IEEE 12207 formats in AY 2000). The standards describe each paragraph in the documentation in terms of content and format, but do not provide an example for each paragraph. Our students have consistently requested examples of the documentation in addition to the standards. This is a difficult tradeoff to address; in some sense, the standards give the students sufficient information to generate the required documentation. It is also difficult to find examples that use the required standards and incorporate tailoring decisions that are appropriate for the course project. In the 2000 AY we began providing examples whether or not they matched the required standards and expected tailoring; unfortunately, student response was that they needed more examples! We revisit this tradeoff every semester, and continually evaluate the alternatives we could use.

One of our most significant tradeoffs in the course is deciding how to balance emphasis on the process of software development and emphasis on the product the students create. On one hand, the software development students have accomplished before this course includes very little, if any, emphasis on process. Given the importance of process in real software development activities, we want to ensure that our students get appropriate exposure to process issues. On the other hand, we also do not want our students to believe that if they follow an appropriate process, it does not matter if they generate a working product! We must therefore also provide the appropriate emphasis on the product the students are developing to ensure it meets the project requirements.

We have not yet perfected our response to this tradeoff. In the 1998 AY, we combined the teams in the middle of the second semester, and the combined team generated a product that met many but not all of the requirements. We planned to have a student complete the product the following semester in an independent study, but satellite problems obviated the need to complete the ground station.

Neither team generated a working product in the 1999 AY. In the 2000 AY, one of the two teams generated a product that met the requirements. The control tower personnel are using the working system as a proof of concept to gain funding for a full-scale system to be used in the USAFA tower as well as other Air Force and FAA sites.

Another tradeoff we face in the course is developing the interactions between the students and the customer. Interacting with real customers is invaluable experience for the students in the course, and other software engineering courses have incorporated extensive procedures to provide these interactions [5]. As in real life, however, the students in the course find that they need to deal with a wide range of customer capabilities.

In many cases, dealing with a knowledgeable customer can be very beneficial for the students (see [10], for example). This can be taken too far, however; one of the customers for the satellite ground station was actually the developer of other ground stations. Discussions of the requirements with this customer often were of the form "This is how I did it, so you should do it this way" rather than "This is what the system needs to do." These interactions were realistic, but frustrating for the students. On the other end of the spectrum, the students working on the UAV simulator ran into some mathematical problems with the formulae used to model the aircraft. The students worked with the customer in this case to try to identify the problem through a step-by-step example, but were unable to identify and correct the problem. This is one of the causes for the student lack of success on that particular project. The air traffic controllers for the CEATMS project were probably the most realistic customers; they were knowledgeable without directing implementation details to the students. Another aspect of realism the controllers provided was their changing requirements; a source of frustration for the students that in fact helped them see how real project requirements evolve.

Another tradeoff to be addressed is the development model to be used in the course. For many years, the waterfall model was the accepted model for software development. There are numerous other models that can be used, however, including the spiral model, incremental development, and others [6]. For the 1998 and 1999 AYs we used an incremental approach similar to that presented in [4]. One of the problems with using this approach from the beginning, though, is that students complete requirements analysis and design activities before they have been covered in sufficient depth in the course. In the 2000 Academic Year, we used the requirements analysis and design portions of the waterfall model in the first semester, then had the students complete multiple iterations of the system in the second semester. The latter combination seemed to be particularly effective, since it lets the instructor cover important requirements analysis and design concepts in sufficient depth before the students actually apply those concepts.

We also recognize that a significant portion of real-world software development consists of modifying existing systems rather than creating new systems from scratch. We have therefore tried numerous approaches for providing the students in the course with some "maintenance" experience. For several years prior to the 1998 AY, we incorporated a maintenance exercise in which the students would modify an existing system to provide additional functional capabilities (similar ideas are presented in [2]). We found, however, that to make this a non-trivial task for the students, the system being modified had to be fairly complex, which in turn led the students to spend an inordinate amount of time trying to understand the system before adding the required capabilities. We therefore considered other options for providing this kind of experience.

The option we selected was to have students change teams between the first and second semester and start from a new project baseline. By having students change teams between semesters, we gave the students the experience of working with code developed by someone else as they continued the development in the second semester. By starting both teams from the same baseline at the beginning of the semester, we facilitated class discussions about the project in the second semester; by the end of the first semester, the two designs for the system were significantly different. When selecting which project baseline to carry forward from the first semester, the instructor tended to select the system with the worst documentation. This also enhanced the realism of the student experience! In the 2000 AY, we did not require students to change teams (based on input from the SMT), but we did require that they change management position and subsystem group, so the students still worked with a design that was developed by a different group of students.

One of the most difficult issues we have faced in the course is evaluating individual contributions to the group project. Evaluating individuals in a team project is clearly not unique to our course, as it must be addressed in any course containing a component of group work, and there are many techniques that can be used to address this issue (see [11], for example). Our technique for assigning individual grades for the project is described below.

The management staff for the project are evaluated based on their performance and the documentation they provide, so assigning individual grades for them is straightforward. The issue arises when we assign individual grades to the members of each subsystem group. We start by evaluating the group's performance and documentation, giving us a "group grade" on which to base the individual grades of the group members. We then ask all the students in the class to rate (with written justification) the other members of their subsystem group on a scale of 0.8 to 1.2. The ratings are evaluated by the instructor and are then used as a multiplier. For example, a group member rated with a 0.8 by all the other group members would receive a grade of 0.8 times the group grade. In this way, the group grade is adjusted based on individual contributions to that grade, both (slightly) punishing those who don't contribute and (slightly) rewarding those who contribute more to the group.

One of the issues we are trying to address with the multiplier is the "freerider syndrome" [3]. Within a group, some students will work harder than others while some will act as "freeriders" and depend on the other group members to do the work for a grade they will then share. By including the multiplier, we attempt to offset a student's ability to get a free ride.

There are still problems with this approach, however. For example, a student who does NO work for the group would still receive at worst 80% of the group grade. While these situations are extremely rare, they have happened in the course. In the 2000 AY, we asked the SMT for feedback on this issue. After soliciting ideas and comments from the other students in the class, they proposed that we retain the current approach of using a 0.8 to 1.2 multiplier.

We have also grappled with the issue of the programming language for the project - should we dictate a particular programming language or let the students select one? In the government it is very common to select the programming language as one of the project requirements, but the students seem to resent being told which language to use. Our response to this tradeoff has been evolving over time; in the 1998 and 1999 Academic Years, we dictated the use of Ada (our "core" language at USAFA) for the project, but in the 2000 AY we let the students select the programming language. We required a Programming Language Selection Study to support their decision to ensure careful thought went into their selection. One of the teams selected Ada, while the other selected Java; an interesting result was that, despite the use of Ada throughout the computer science curriculum, the team using Java was the team that developed a working system.

5 Conclusions

Developing a software engineering course or capstone sequence that provides students with realistic software development experience and strikes a balance between technical and management issues can be difficult. In addition, equitably assessing each student and addressing the other tradeoffs in the course is a complex challenge.

In this paper, we have described the projects used at USAFA over the past three years, the course structure we use in the capstone sequence, and the tradeoffs we have identified and addressed. Our approach continues to evolve, of course; we continually evaluate our responses to important issues and determine whether or not changing selected aspects of the course could address those issues more effectively. We believe a key component of this course evolution is student involvement, and using a Student Management Team to help gather student feedback appears to be a unique and effective methodology.

References

[1] Bruegge, Bernd, Cheng, John, and Shaw, Mary. A software engineering project course with a real client. Technical Report CMU/SEI-91-EM-4, Carnegie-Mellon University Software Engineering Institute, 1991.

[2] Churcher, Neville and Cockburn, Andy. An immersion model for software engineering projects. In *Proceedings of the Second Australasian Conference Computer Science Education*, Sydney, Australia, July 1996.

[3] Daft, Richard and Marcic, Dorothy, *Understanding Management*. Second Edition, Harcourt Brace & Company, Orlando, FL, 1998.

[4] Hutchens, David H. and Katz, Elizabeth E. Using iterative enhancement in undergraduate software engineering courses. In *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*, Philadelphia, Pennsylvania, February 1996.

[5] Polack-Wahl, Jennifer A. Incorporating the client's role in a software engineering course. In *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, New Orleans, Louisiana, February 1999.

[6] Pressman, Roger, *Software Engineering: A Practitioner's Approach*. Fourth Edition, McGraw-Hill, New York, NY, 1997.

[7] Robillard, Pierre N. Teaching software engineering through a project-oriented course. In *Proceedings of the Ninth Conference on Software Engineering Education*, pages 29-39, Daytona Beach FL, April 1996.

[8] Song, Ki-sang. Teaching software engineering through real-life projects to bridge school and industry. *SIGCSE Bulletin*, 28(4):59-64, December 1996.

[9] Tomayko, James E. Teaching a project-intensive introduction to software engineering. Technical Report CMU/SEI-87-TR-20, Carnegie-Mellon University Software Engineering Institute, 1987.

[10] Villarreal, E.E. and Butler, Dennis. Giving computer science students a real-world experience. In *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education*, Atlanta, Georgia, February 1998.

[11] Wilkins, Dawn E. and Lawhead, Pamela B. Evaluating individuals in team projects. In *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education*, Austin, Texas, February 2000.