Learning Software Engineering by Doing: Progress Report on a Capstone Sequence Involving Student Managed Teams

Clark Savage Turner, Gene Fisher, Daniel Stearns
Department of Computer Science
Cal Poly State University, San Luis Obispo, CA 93407
Email: {csturner, gfisher, dstearns}@calpoly.edu

Abstract

Cal Poly recently instituted a software engineering major, which includes a yearlong capstone project course. The main goals include:

- emphasis on a team approach to software development
- focus on software process
- use of a real-world project
- the opportunity for seniors to develop technical leadership skills

In order to meet these goals, the sophomore-level software engineering courses are aligned with senior-level courses. The lectures run independently with appropriate focus, but the labs are coordinated so that the seniors provide technical leadership, mentoring and responsibility for deliverables for the teams of sophomores.

In the first two years, industrial partners sponsored the project. The fall quarter began with a senior-level course in requirements engineering: the seniors developed the requirements specification document with the industrial partners. During the following two quarters the sophomore teams would design, construct, and deploy a product to those specifications under the management of the seniors.

Difficulties in course implementation for the first two years led to analysis of student, faculty and industrial partner feedback. Due to extreme coordination difficulties, a university project replaced the external project and industrial partner. Further, the sophomores were unprepared to participate in software construction without domain knowledge, so the courses were realigned so that the sophomores would be involved in the requirements engineering and construction (but not deployment).

This paper provides a progress report for the third year (2003-2004) of Cal Poly's Capstone Sequence and preliminary analysis of changes and effects.

Introduction

California Polytechnic State University (Cal Poly) was founded a century ago with a special mission -- knowledge is best gained through the application of theory to practical problems. The Cal Poly motto "*Learn by doing*" is widely quoted and permeates all degree programs. Cal Poly

hosts several nationally-ranked engineering programs and graduates a thousand engineers per year. Each graduate completes an extensive general education program combined with study of a specific engineering discipline. Graduates enter their careers with a practical understanding of their profession due to a series of lab and real-life experiences that apply theory to real engineering problems.

The Computer Science Department resides in the College of Engineering at Cal Poly. It hosts three undergraduate majors – Computer Science, Computer Engineering and now the Bachelor of Science in Software Engineering (SE). All three majors prepare students for professional careers in software and hardware development. A large majority of the graduates enter the workplace immediately after graduation.

The Capstone Project Concept

The proposed SE curriculum complements the existing computer science program emphasis on a solid base of concepts and technology skills with an introduction to resource and technical management. The challenge has been to fit these additional learning units into a four-year program without sacrificing other valuable requirements. The challenge is addressed through two tightly interwoven sequences of courses. An initial two-course sequence is offered in the second year to introduce software engineering principles while students construct a sponsored software product. In the fourth year, in a three-course Capstone Sequence, students study advanced software engineering and lead a team to develop a software product. This sequence comprises three courses taken consecutively in a single academic year. For the purposes of this paper, the three capstone courses are designated as follows:

1. Requirements Elicitation (402): In this course, the students elicit requirements from the users and write a software requirements specification. The course content includes formal specification writing, requirements modeling, rapid prototyping, and elicitation techniques.
2. Software Construction (405): In this course, the students build the initial version of the software product and deploy that version at the customer's site. The course content includes design modeling, software construction techniques, software quality assurance, and software project management.
3. Software Deployment (406): The students maintain the product during this course. They add functionality to the product, repair defects, create variants and perform usability testing. The course involves release management, software maintenance, deployment practices, software quality metrics, and metric-based process improvements.

See Figure 1 and Table I for information regarding these two sequences and their alignment with the sophomore level courses.

This is the third report regarding the Capstone projects. It is provided to illustrate the more critical questions raised, to examine the tradeoffs made, and to analyze results so far.

Overview of paper structure

This report starts with a brief history of past Capstone Projects. The current project and its relevant details are then given. The Requirements Elicitation Course is then examined followed by the *Software Construction* Course. Analysis and thoughts about future changes in the structure and goals of the courses are provided at the end of the paper.

A Short History of the Cal Poly Software Engineering Capstone Project

To date, the Capstone Sequence has run three times. The Software *Requirements Elicitation* Course ran independently the first and second times. The next two courses, *Software Construction* and *Software Deployment* were aligned with the sophomore *Intro to SE I* and *Intro to SE II,* respectively, so that the senior SE students performed management duties and were responsible for the deliverables for these three different projects.

While both the faculty and the students learned a lot of software engineering and deemed the whole sequence a success, the following concerns arose:

- A lack of strong commitment from the industrial partners diminished the students' learning experience.
- The esoteric nature of the projects distracted the students from the software engineering challenges.
- Necessary tools and working environments had a steep learning curve and were distracting to the students.

On that basis, the decision was made to take on only one industrial partner who offered a strong commitment to follow through with the project. Further, the project would be chosen so that the domain was not so difficult for the students and the necessary tools and environments were already familiar to them. See generally [1].

During the second run of the *Capstone Sequence*, the changes were implemented. One industrial partner was chosen; their project appeared to be reasonable in scope and their commitment strong. The sequence again provided a generally successful industrial software engineering experience. Efforts to analyze and improve noted the following difficulties:

- Intellectual property issues involved significant faculty time during the first course in *Requirements Elicitation*.
- If the customer was busy or missed a deadline, all teams experienced delays and frustration.
- The single project resulted in competition for scarce domain materials that should have been shared openly among teams.
- Customer meetings were difficult to arrange and manage. When held, they were large and unwieldy.
- The domain itself was still difficult for the sophomores: they needed several weeks of their 10 week *Intro to SE I* course just to gain working knowledge and begin design.

These observations resulted in two changes to the *Capstone Sequence*:

- No corporate project – incorporate a University project of sufficient scope with local users.
- Course realignment: the sophomore *Intro to SE I* students would now be involved in the *Requirements Elicitation* course and managed by the seniors in the Requirements elicitation process. The *Intro to SE II* students would then continue with the *Software Construction* course but not with *Software Deployment*, the third course of the sequence. See generally [2].

Software Engineering Capstone Structure, 2004

The structure of the current sequence reflects the changes made over the previous two. The aligned courses share a common laboratory space and time so that the Capstone student managers have scheduled blocks of time to manage their *Intro to SE* teams. The current structure is given below in Figure 1. Further background in the individual courses is given in Table 1 below that.

Figure 1: Capstone Course Sequence and Alignment

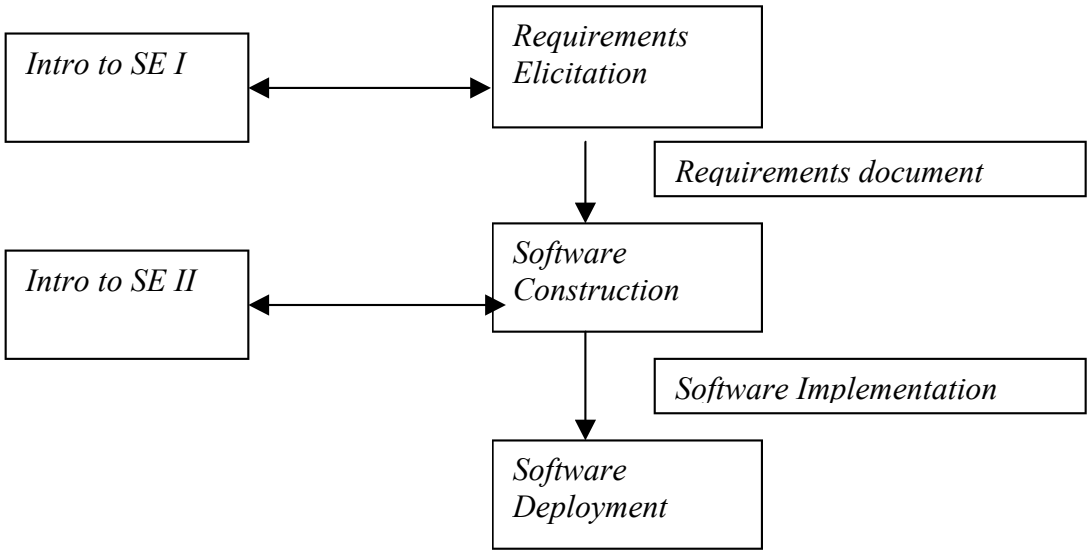

TABLE 1: SOFTWARE ENGINEERING COURSES IN THE CAPSTONE

Course	Abbrev used in text (in italics)
CSC 205: Introduction to Software Engineering I	*Intro SE I*
CSC 206: Introduction to Software Engineering II	*Intro SE II*
CSC 402: Software Requirements Elicitation	*Requirements*
CSC 405: Software Construction	*Construction*
CSC 406: Software Deployment	*Deployment*

Project Description in 2003-2004

There are 10 teams involved in the realigned pairs of courses; each team has selected one of the following three projects to complete. All three projects have similar attributes: the domain is reasonable for sophomores and the main user is a Software Engineering faculty member.

Electronic Classroom Description (Eclass)

An ideal lecture medium allows the sharing of information between instructor and students. Few lectures fulfill this ideal. Time and resources are used in photocopying lecture material, handing out papers, and taking attendance. Instructors use the chalkboard to write down important topics and notes. Students copy the information and add their personal notes.

Eclass allows the instructor to display lecture material and notes as he wishes on a classroom overhead display. It differs from a traditional lecture presentation because the material is simultaneously available on each student's classroom machine and can be made available remotely or for later use. The student need not copy anything and can add his own notes with ease. In addition, the student can interact with the instructor and the lecture: she may write or draw something that is displayed to all of the students in the room, in the style of an electronic whiteboard.

Grader Tool Description (Gradepad)

Many teachers spend time organizing and recording grades and assignments. With a spreadsheet GUI, Gradepad allows teachers to enter grades into a class roster downloaded from a campus database. Teachers can peruse grade categories such as "Tests" and "Quizzes," and they can also zoom in on a category to check student scores on individual questions.

Gradepad also calculates a variety of statistics and displays them graphically. Teachers can graphically adjust pie charts and histograms to adjust their actual grade distribution. Finally, they

may configure a late policy based on grace days and decay to automatically deduct points for late assignments.

Students can also use Gradepad - to display their grades and pose grade projection scenarios. The grade projection feature allows a student to forecast her final grade based upon predictions for future assignments. Also, the students may submit assignments electronically to Gradepad and receive a record of the submission.

Examination Tool Description (Testtool)

Testtool allows a professor to write and store questions in a central database and to use those questions to automatically generate examinations, using a variety of selectable criteria. The examinations can be automatically graded by Testtool if desired.

Testool allows a proctor to administer a test in a computer lab setting that gives students a comfortable environment. If the test is graded by Testtool, students get immediate feedback on their examination.

Software Requirements Elicitation Course (CSC 402)

Intro to SE I Teams

Since the courses were realigned, there were teams of 4 to 6 sophomores managed by the senior *Requirements Elicitation* students. Each team worked on one of the projects. Three *Intro* sections were taught, so each of the three projects had three teams assigned to it.

A single set of lecture slides and materials were used for all three *Intro* sections taught by the two instructors to promote consistency among the students on the teams to be managed. The process used was fairly formal, the RSL specification language [3] was used, and student teams were expected to follow standard operating procedures. For more insight into the process and materials used, the class webpages may be viewed at waldorf.csc.calpoly.edu/~gfisher under the current CSC 205 and CSC 402 pages.

User Interaction

Though users outside the software engineering faculty were identified and participated to some extent, the student teams had biweekly formal meetings with one or both instructors and came to rely on those meetings for requirements elicitation activities.

Course Objectives and Evaluation

At the outset of the *Requirements Elicitation* course in Fall 2003, the objectives were:

1. To develop a requirements specification for three SE faculty software products.
2. To build a rapid prototype of the products.
3. To learn about software project leadership.

4. To reinforce the learning objectives of *Intro to SE I*.

To achieve the first objective, the students were initially given a two-fold job description:

- to lead *Intro to SE I* students in their project work, which entails writing the more straightforward parts of the requirements specification;
- to develop the advanced parts of the requirements specification, and to do the rapid prototyping.

As the quarter evolved, this job description was revised to allow the *Requirements Elicitation* students to choose how much actual development to do themselves. They were told that their course performance evaluations would be "product based", that is, they would be graded on the objective quality of the produced product, versus subjectively on their leadership abilities. Hence, it was up to them to choose how much development work to assign to themselves and to the *Intro* students.

By the end of the term, only three of the twenty-five *Requirements Elicitation* students had done any substantial work on the requirements specification themselves. The other twenty-two students had focused on leading the *Intro* students in their work. For those teams who produced a product of reasonable quality, the leaders did spend a considerable amount of time reviewing and editing the Intro students' work, but the Intro students were directly responsible for the delivered work products.

This style of work appears to have led to an inferior-quality work product compared to one where the *Requirements Elicitation* students have done more of the work themselves. In general, the first course objective, that of a production-quality requirements specification, was far too ambitious. It was not achievable given the other course objectives.

Regarding the second course objective, that of a rapid prototype, only three of the nine project teams produced a reasonable result. Postmortem evaluation of this result is again that this objective was too ambitious given the other objectives.

The third objective on project leadership was very well achieved. All *Requirements Elicitation* students were required to assume leadership duties. As is of course expected, some performed better than others, however all learned a great deal from their leadership experiences.

The last course objective was to have *Requirements Elicitation* students learn more about the technical material of CSC 205. That is, *Requirements Elicitation* students were to learn more about writing functional requirements, specifying requirements models, testing requirements specifications, and project configuration management. To the extent that managing these activities constitutes "learning more" about them, this goal was achieved. However, at the outset of the term, the instructor had some more specific technical learning objectives in mind such as: requirements specification for distributed systems, specification testing via formal model checking, and advanced aspects of version control using CVS (the Concurrent Version-Control System). Given the evolution of the course to a predominant focus on leadership and management, the objectives for advanced technical learning were not achieved.

At the end of the term, the *Requirements Elicitation* students were asked to comment on their experiences in the class, and to comment on how the course could be improved in the future. In particular, the students were asked to give answers to the following topic-related questions:

- Leadership Experience -- should it be required for all *Requirements Elicitation* students, for some, or for none?
- Deployable Product -- should the product be production-quality, lesser quality, or a "toy"?
- Real-World Customers -- should outside customers be used, inside (campus) customers, or customers "simulated" by the instructional staff?
- Pairing with *Intro to SE I* -- should the *Requirements Elicitation* class be paired with *Intro to SE I*, or not paired?
- Course Focus -- should the focus of the *Requirements Elicitation* course be on requirements techniques, project management, or both?

A summary of student responses to these questions is presented in the conclusions.

Software Construction Course (CSC 405)

The *Software Construction* course (405) was aligned with the sophomore level (206) course, primarily the same students that took the Intro to SE I course and wrote the requirements. Most of the *Construction* course students served as leader of a team charged to construct the products from the requirements. The project portion of this course had a simple objective - build a product that could be deployed to the software engineering faculty.

The *Construction* course has several objectives revolving around the design and implementation of a large project:

(1) to learn the principles of software design and apply them to the project;
(2) to learn how to work on a team (typically 8-12 students);
(3) to learn the fundamentals of software testing and apply them to the project;
(4) to learn and apply software project management skills; and,
(5) to learn and apply Software Quality Assurance practices.

<u>Project Scale and Manageability</u>

Three software projects were selected for the *Capstone Sequence*; each project was of interest to the Software Engineering faculty. As decided after the last sequence ran, there was no industrial partner involvement although several users outside the SE faculty were identified. Those users contributed slightly, only during the requirements elicitation phase. For all practical purposes, the students viewed their course instructions as the user.

There is a substantive difference between a project built for an industrial partner and a project controlled by a course instructor. Observations and discussions with students are clear. A project for an industrial partner creates more difficulties for the students because:

1. The industrial partner, like all users, is constantly changing his mind and creating new requirements.
2. The industrial partner expects the delivery of a working product. (Even though much effort may be spent to lower such expectations.)
3. The industrial partner is much more difficult to contact than the course instructor.

Leader grades

A critical decision was made when the capstone courses were designed. The leader's grade would not be based on the successful completion of the project; to do so would encourage the leaders to simply do the technical work themselves. The leader's role needs to be: educator, mentor, project planner, reviewer, manager. Consequently, the leader's grade on the project is assigned similar to the corporate world. It is subjectively based on their success in leading their team. It is quite possible for a team to build an excellent product with a poor leader and vice versa. The leadership qualities of each leader are also evaluated by her team; this evaluation can possibly have an effect on the grade.

Intro to SE II student grades

Multiple instructors teach the sophomore students. Each instructor designs her own grading scheme; no attempt has ever been made to design a common grading scheme for all instructors. There is a general policy that students are assigned individual grades; a team grade, applied to all team members, is considered unacceptable for the obvious reasons.

The two instructors in the 2003-04 *Capstone Sequence* used two different grading schemes.

1. One instructor delegated grading authority to the leaders. Each leader was required to do a formal evaluation of each team member and assign an appropriate project grade.

2. The other instructor did not delegate grading authority to the leaders. Rather, the leaders were given "veto authority" over the project deliverables. Once the leader accepted a deliverable, the instructor assigned individual grades in the traditional way.

Software Deployment (406)

This paper was written before the Software Deployment course started but a few observations are in order. There will be no sophomore students working with the leaders; the leaders will be learning deployment and maintenance principles alone. The leaders will, of course, be working on the products that were created in the previous two teams. The lack of sophomore students means the course will cover its topics (deployment issues, software maintenance, configuration techniques, installation practices, marketing, technical support) with a stronger emphasis on theory.

As this paper is written, the leaders have already become aware that they will, personally, be maintaining the code base created by their teams. This awareness is manifesting as a stronger focus on the code quality in the teams. The leaders clearly want high-quality code to work on!

Changes from Last Year and Results

2002-2003 versus 2003-2004 *Capstone Sequence*

After the previous Capstone Sequence, 2 changes were implemented:

1. no corporate project: take on an internal project to simplify logistics and reduce domain complexity; and,
2. course realignment: get the sophomores involved in the Requirements elicitation phase to increase their domain comfort.

No Corporate Project

As desired, the problems in negotiating a reasonable Intellectual Property agreement (with its attendant difficulties) were eliminated. This was a welcome result.

However, the other implications of the change were not as simple. The 2002-03 capstone students specified, designed and constructed a project for an industrial partner – Brocade Communications. The 2003-04 students worked on projects for the Software Engineering and Computer Science faculty. There was a wide gulf between the two domains and a substantive difference in results.

Comparison of Domains – Corporate (Brocade) Project

Brocade's primary product line is Storage Area Networks (SANs). Brocade provided an actual SAN for the students at Cal Poly and provided training on the hardware/software components. At the halfway point that year, few of the students (leaders included) were comfortable with the SAN. Only two students could deal with technical problems. Almost all of the sophomore students were uncomfortable logging on the SAN and running its tools.

The Brocade project was to construct an automated tool to generate and execute a set of SAN test cases. For a simple SAN configuration, the number of test cases reached 6 figures. CAPATT required the use of several 3rd party libraries and tools provided by Brocade and a SAN trade group. At the halfway point, all of the teams and most of the students were in a constant wrestling match with these tools. In addition, there was a major new release of the 3rd party tools in the middle of the *Construction* course.

At the halfway point, none of the 16 teams had delivered a required prototype. The prototype was intended to demonstrate the team architecture – interface with the 3rd party libraries, connect with the SAN and generally prove the technical feasibility of the project. The teams were spending most of their time understanding the requirements, dealing with changing requirements and learning the SAN domain and its tools.

Comparison with Software Engineering Faculty Projects

As this is written, the halfway point this year, the project teams have demonstrated their prototypes. All of the teams showed working software of good quality; some of the teams demonstrated a prototype that is nearly a finished product.

There is no industrial partner; the Software Engineering faculty serve as the users. There are no 3rd party tools or libraries. There is no hardware, other than the platform to execute the programs. Every student understands the domains well enough to participate in domain discussions.

Course Realignment

The realignment of the courses was proposed so that the sophomores would be involved in the domain and gain enough knowledge to significantly contribute to the construction of the product. This change was difficult to evaluate because the domain chosen for the faculty project was so familiar to the students that they were already domain experts. The sophomores were, indeed, not slowed down by lack of knowledge of a complex domain in the current project. The realignment appears to be a good idea, especially if a more complex or unfamiliar project domain is undertaken in the future.

Conclusions and Future Directions

The clearest trends from the student responses to the questions outlined previously are the following:

1. Leadership experience should be required for at least some *Requirements Elicitation* students.
2. Products should be less than production-quality.
3. Customers should be on-campus or simulated, but not real-world.
4. The course focus should be on both management and requirements techniques.
5. There is no clear conclusion regarding the course pairing,

The authors are largely in agreement with these student responses. The following additional observations are relevant:

1. The authors believe that leadership experience must be required for SE majors, and can be optional for other majors taking the course.
2. Products must be deployable in some form. The deployability will be stressed at the outset, but may be relaxed later in the term to a minimum deployable subset of the initial requirements.
3. Unless customers and student teams are carefully selected, using an industrial partner is not considered practical. In a required class, perhaps one team and one customer can be so selected.
4. Since management experience is essential for SE majors, an unpaired version of the course would require either that the SE majors manage themselves, or separate management experience would be gained in a course independent of the Capstone Sequence.

5. Instructors have mixed feelings about the course focus. There are plans for future offerings that focus predominantly on management or predominantly on requirements techniques and modeling.

References

[1] Stearns, Meldal, Turner, "Ten Pounds in a Five Pound Sack: Providing Undergraduate Software Engineering Students with Technical Management Experience," Proceedings of the International Conference on Engineering Education, August, 2001, Oslo, Norway.

[2] Stearns, Dalbey, Turner, Kearns, "Report: A Capstone Project involving a Hundred Students, for an Industrial Partner," Proceedings of the International Conference on Engineering Education, July, 2003, Valencia, Spain.

[3] Fisher, Gene, "The RSL Reference Manual"
http://waldorf.csc.calpoly.edu/~gfisher/classes/205/doc/ref-man/

Author Information

CLARK SAVAGE TURNER, J.D., Ph.D., is an Associate Professor of Computer Science at Cal Poly State University in San Luis Obispo, CA.
GENE FISHER, Ph.D., is a Professor of Computer Science at Cal Poly State University.
DANIEL STEARNS, M.S., is an Associate Professor of Computer Science at Cal Poly State University.