Ten Pounds in a Five Pound Sack: Providing Undergraduate Software Engineering Students with Technical Management Experience

Daniel Stearns[1], Sigurd Meldal[2], Clark Savage Turner[3]

Abstract *— When Cal Poly created an undergraduate software engineering program, the challenge was to package the "ten pounds" of science, computer science and engineering into the "five pound" capacity of a bachelor's degree. On top of this, we needed to add preparation for the management roles required of a professional software engineer.*

Cal Poly (California Polytechnic State University) was established a century ago with the motto "Learn by doing!" Cal Poly's educational mission requires instructors to apply theoretical knowledge to practical problems. This was the guideline when the Cal Poly Computer Science Department defined a new software engineering program: to provide industry with high-caliber, deployment-oriented software professionals.

A major program objective is "hands-on" experience. The curriculum is founded on traditional computer science, but distinguishes itself by:

- *Emphasizing a team approach to building software and providing leadership opportunities for each student.*
- *Focusing on software process.*
- *Including engineering and management areas such as project planning, resource allocation, quality assurance, testing, metrics, maintenance, configuration management, and personnel management.*
- *Placing a stronger emphasis on mathematics and the use of engineering methods in software design.*

The solution to the packaging challenge: teach a sequence of interlocking software engineering courses: sophomores participate with seniors in a yearlong, team-based project that constructs sizable (> 200 function point) software systems for industrial clients. Seniors participate as technical managers and mentors who assume leadership responsibilities for project deliverables.

This paper describes the experience and lessons learned during our first year.

Index Terms ¾ Software engineering, Undergraduate education, Technical management education, Project-oriented education.

BACKGROUND

Cal Poly and Its Instructional Philosophy

The California Polytechnic State University (*Cal Poly*) was founded a century ago, and (according to anecdotal evidence) its instructional mission was determined by the founder's personal experience during hard times – theoretical knowledge is of limited value unless it's applied to practical challenges.

The belief that theoretical knowledge is reinforced through its application to practical problems governs Cal Poly's instructional philosophy. The motto "*Learn by doing!*" is widely displayed and permeates all degree programs.

Cal Poly is a *polytechnic* institution with several nationally ranked engineering programs that graduate a thousand engineers each year. Each graduate has completed an extensive general education program combined with an in-depth 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 Cal Poly Computer Science Department hosts two undergraduate degree programs – Computer Science and Computer Engineering (co-hosted with the Electrical Engineering Department). The degree programs prepare students for professional careers in software and hardware development. A large majority of the graduates enter the workplace rather than undertaking graduate studies.

In crafting its 3rd program (Computer Software Engineering), the department traded some theoretical topics for an in-depth coverage of software design and other software process topics. The objective is a student with a "*can-do*" attitude and a thorough professional preparation who will live up to high employer expectations.

[1] Daniel J. Stearns, Computer Science Department, California Polytechnic State University, CA 93407, dstearns@calpoly.edu
[2] Sigurd Meldal, Computer Science Department, California Polytechnic State University, CA 93407, smeldal@calpoly.edu
[3] Clark S. Turner, Computer Science Department, California Polytechnic State University, CA 93407, csturner@calpoly.edu

Computer Software Engineering

There is a growing realization that the demands of our profession require fundamental engineering skills. We believe the time has arrived to create an undergraduate software engineering program.

The term *software engineering* dates back to 1972 as was coined by F. L. Bauer:

> "*The establishment and use of sound engineering principles (methods) in order to obtain economically software that is reliable and works on real machines.*" [1]

The transformation of the software profession from a craft to an engineering discipline is an on-going project, and far from complete. The pioneering work of the Software Engineering Institute [6], the subsequent work on a software engineering body of knowledge [8] and an associated set of curriculum guidelines [5] provided us the impetus to create a bachelor's degree in Computer Software Engineering.

The distinguishing features of a software engineering degree arise from the observation that software engineering deals with *deployment-oriented software development.* This phrase establishes what software engineering is about:

- Development of a product to address a real need.
- Timely and predictable delivery.
- Affordability.
- Customer satisfaction.

When measured against the above success criteria, the current state of the profession falls short. An improved education addressing these needs is warranted.

The proposed Cal Poly computer software engineering curriculum (CSE) complements the existing computer science program emphasis on a solid base of concepts and technology skills with an introduction to resource and technical management. We understand the legitimate concern about the depth of knowledge a university can provide within existing curricula. Our challenge is to fit these additional learning units into a four-year program without sacrificing other valuable requirements.

This challenge is addressed through two tightly interwoven sequences of courses. The first sequence is offered during the sophomore year and introduces students to software engineering. The second sequence offers a yearlong capstone software development experience in the senior year. This report describes our first year of experience with the capstone course sequence.

CAPSTONE COURSE SEQUENCE OBJECTIVES

The capstone sequence spans three quarters – a full academic year. The courses making up the sequence are named *Software Requirements Engineering, Software Construction* and *Software Deployment*.

Before entering the capstone year the students acquired foundations in mathematics and computer science. They also completed two courses in manufacturing engineering and a course in organizational behavior, with an emphasis on the dynamics of team collaboration. In the software engineering area, the students have completed a two-quarter introductory sequence, as well as a one-quarter course in individual software development, where they complete a significant individual software construction project. We found the latter important, since the computer science introductory courses do not provide the students with sufficient experience and understanding to support major software development efforts.

The objectives of the introductory sequence are:

- To understand software engineering fundamentals, especially software design and development prior to the deployment phase.
- To achieve basic literacy with the standard software engineering artifacts: requirements specifications, software architecture, design, user interface storyboards, prototypes, and quality assurance plans.
- To understand the notion of software quality attributes and how they flavor the development processes.
- To gain experience with an organizational framework where documents written by others form the basis for their development.
- To engage in significant team-based projects.

We have found that sophomore students are insufficiently prepared and professionally immature to benefit significantly from being asked to manage a software development process. However, we do want the students to get a practical understanding of the challenges of software construction. Therefore, our design of the capstone sequence has the following objectives:

- To provide the students with a thorough understanding – through experience, reading and lectures – of the software engineering processes. We explicitly included post-deployment activities.
- To provide the students with the tools and concepts necessary to successfully schedule software engineering processes, given limited resources.
- To provide the students with an opportunity to learn and practice management skills.

Given the needs of the sophomore sequence as well as the limited number of course units an undergraduate degree provides, we wove together the sophomore and capstone course sequences.

CAPSTONE COURSE STRUCTURE

The software engineering capstone course sequence comprises three courses taken during an academic year. For the purposes of this paper, the three courses are designated as:

1. Requirements
   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. Construction
   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. Deployment
   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.

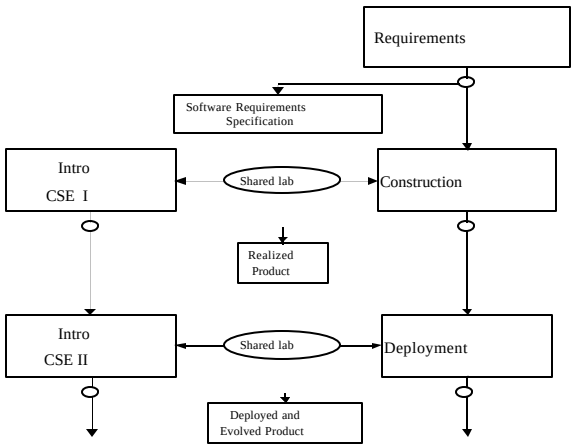

Figure 1 Capstone course sequence

To offer students realistic leadership roles, the Construction and Deployment course laboratories are combined with the laboratories of the Introduction to Software Engineering courses. Each student team contains students from two courses. The Construction course students serve as leaders to the Intro CSE I students. The Deployment course students work with and serve as leaders to the Intro CSE II students. This unique feature of Cal Poly's capstone sequence created a number of human interactions as the students from two courses worked on software projects. Figure 1 shows the relationships among the 5 courses.

PARTICIPANTS

The capstone course sequence stands or falls on the shoulders of three participant groups: faculty, students and industry project providers. In this section, we describe some of our choices and the reasoning we used in making choices.

Faculty

The capstone sequence is based on a yearlong interaction with industry through real-world projects. We needed, and obtained, academically qualified faculty with significant industrial experience. As a new, experimental course sequence involving industry interaction, extra work was required and the risks were unknown.

Students

For the most part, only motivated, highly qualified students requested the courses. Some qualified students could not be accommodated due to the limitation of class size at thirty. For the capstone sequence courses, our choice of students was based on grades in the introductory software engineering courses, level of industrial experience and faculty recommendations.

An exception was made for the students from St. Jude Medical. In that case, the project provider selected the students (their own employees) to work on their project. Their employees were admitted as normal matriculated students in the Cal Poly Computer Science department.

There was no selection process used for the introductory courses; those courses are required for all students. One section was selected to work with the capstone sequence; the students were informed the first day of class and given the opportunity to change to a "normal" introductory section - every student chose to remain with the experiment.

Industry Project Providers

In order to make the capstone sequence work, we needed a complete instructional package from our industrial supporters. In particular, we needed:

- Real industrial projects with appropriate characteristics.
- Funding to support our instructional mission.

We emphasized our instructional mission to our project providers! We stressed that there would be no deliverable guarantees. At best, we expected to deploy a working beta version with significant functionality.

Why would our project providers lend time, financial support, and corporate proprietary information to Cal Poly

student projects? Perhaps they believed our students would generate valuable solutions? This was never considered likely. [4] The primary motivation was recruiting: industry providers interact with our best students. These students became familiar with the industry provider. Recruitment costs and risks are vastly reduced by such engagement.

St. Jude Medical had another motivation for their involvement. They sought to increase employee retention by sponsoring employee professional development.

Project Expectations

Our expectations for the projects were driven more by the "real world" target than a purely academic inclusion of certain elements. We could be flexible in accepting projects as long as we could envision all of the following elements as a natural extension of development:

- GUI interface
- Database component
- Multi-threading
- Several distinct components
- Rich in use-cases and business logic
- Real nonfunctional requirements
- Scoped for a three course sequence

After meeting with several industry supporters, we chose the three projects we felt had the best potential to support our goals. Each project started as someone's "pet" idea that could result in a prototype of some useful software not currently considered a priority by the provider.

Three companies, Airtreks, IPTech, and Saint Jude Medical were our industry providers during this first year. Time, space and nondisclosure agreements prevent discussion of project details, but each one involved unique and interesting problems that potentially met our criteria.

We further note that we could adjust the scope of the projects during course evolution. We could add important elements dynamically, just like the real world. Of course, we could also help teams prune the project scope to meet actual time and resource constraints.

[4] There were delicate intellectual property issues to be worked out here. The university usually claims property rights to student and faculty creations. The project providers would naturally want those rights. Further, the project provider necessarily shares proprietary information with faculty and students to get the project started. We explained to the university that our mission was uniquely instructional and that no valuable creations were likely. The university accepted that position and waived rights to intellectual property created during the capstone courses. Industry providers required students and faculty to agree to nondisclosure of corporate proprietary information. We considered this a workable solution.

HOW DID IT GO? WHAT DID WE LEARN?

The Requirements course students, working in teams, wrote Software Requirements Specifications that were approved by each of the companies. Each student team devoted the entire course to one of the three companies. Students in the subsequent courses used the software requirements specifications to define their course objectives.

Requirements Course

The Requirements course produced three usable specifications and the students learned how to elicit complex requirements on real projects. For those two reasons alone, the Requirements course was deemed a success. The company liaison representatives worked closely with the student teams. St. Jude Medical and IP Tech had company employees enrolled in the course. One of the capstone sequence instructors had a professional connection with the Airtreks. These connections were considered important, but not vital, to the success of the course.

Students in the Construction and Deployment courses used the SRS documents as a baseline document but they experienced a number of problems:

- There were a number of inconsistencies and incomplete sections in the software requirements specifications. While such problems are normal and provide an excellent learning experience, they cause real difficulties for developers on a tight schedule. Construction course students spent too much time revisiting the requirements issues.
- The Requirements course students believed that requirements elicitation meant 'ask the customer what he wants'. In all 3 projects, the customer didn't have a clear or feasible system vision.
- Requirements course students were learning how to write product quality specifications. In general, they did an excellent job but somehow came to believe that SRS quality = f (SRS weight).
- Two of the projects were from complex problem domains that required significant effort to understand. Lack of domain knowledge is a problem that student projects can do without.
- The Requirements course students had little ability to produce accurate estimates. Function Points were taught in the Requirements course. But even if the student estimates were accurate (they weren't), there is no evidence to show that function point estimates apply to a college course.

Construction Course

The Construction course had a clearly stated objective: build the software specified by the Requirements course students. This objective was complicated by:

- The student turnover from the Requirements course was 50%. Many new students joined the sequence; once the CSE program is in place, this won't happen because every student will take the entire sequence.
- A major part of the capstone sequence is a required leadership experience. The Intro CSE I students joined the course and were assigned to student teams. Each student team worked on one of three projects for the entire course. Each team included Construction course students and Intro CSE I students.

Most software engineering students are delighted to enroll in a course where they construct software. They typically believe that software construction is the ultimate job skill and most students enjoy it. But the Construction students quickly learned the difference between 'programming for fun' and 'programming for real.' They had to deal with unclear and incomplete specifications and customers who constantly change their vision. They quickly understood the immediacy of a 10 week schedule for a large (200 function points) project.

The capstone course instructors were naturally delighted with all these revelations because they provided wonderful fodder for learning software engineering. In fact, the students often forget, and had to be reminded, that they were working on a project as part of a university course.

In any case, course expectations were set quite high and the pressures mounted as the students realized the realities of completing a real project. Each student team appointed a product manager to deal with requirements issues. The product managers juggled the customer requirements, the Requirements course specifications and the scheduling realities.

We expected team issues to dominate the Construction course and were not disappointed. Each Construction course student accepted a leadership role; some accepted minor multi-person task assignments. Others managed major development or project tasks. But each Construction student practiced leadership; a major component of their course grade depended on leadership activities. There were several problems/issues caused by this structure.

- The Construction course students had responsibility without authority; the only real authority in a university course is the grade. This problem was discussed constantly throughout the course; in the end, we decided that the leadership students would write formal, weekly evaluations of the Intro CSE I students. The Intro CSE I instructor used these evaluations as a major component of each student's grade.
- In any group, there are people who are unwilling or unable to exercise leadership skills. It is difficult to force leadership on someone who is quite uncomfortable in that role. However, some Construction students who had never acted as a leader in any capacity, grew markedly during the course.
- In many cases, Intro CSE I students were technically stronger than their leaders. Even though such a relationship is common in the corporate world, students had a difficult time adjusting to it.
- The Intro CSE I students were given a time budget for their work in the course (15 hours per week - a typical expectation for a 4 credit course). The Construction course leaders paid little or no attention to this limitation and treated the Intro CSE I students as employees. This was one manifestation of a number of time management problems. University students live a complex life full of activity and can't be expected to completely devote themselves to a project in one course.

Deployment Course

The Deployment course was designed to instruct students on the myriad of issues involved ***after*** a software product is deployed. The Construction course students created three software products that were installed at customer sites. In each case, the customer installed and accepted the product. Notably, all three customers failed to invest much energy in the installed product; this diminished the Deployment course experience for the students. What happened?

- The St. Jude Medical students (full-time employees, part-time students) lost management support for their time spent in the courses. The students' work projects took priority; in the end, the students withdrew from the capstone sequence.
- The IP Tech interest was never particularly strong; their employees who were enrolled in the courses continued to represent the company interest without much support.
- The Airtreks project, while of long-term interest to the company, had little immediate relevance.

The lesson learned: get strong commitments from the corporate sponsors with identified individuals who will remain invested in the capstone projects for the entire year.

We decided to replace the team organization in the Requirements/Construction courses with a completely new structure, matrix management. The Deployment course focused on release and maintenance issues that don't require a team structure. In addition, the instructors wanted to test a

completely different relationship between the two linked courses (in this case, Deployment and Intro CSE II).

The matrix management organization was designed to provide technical leadership tasks for each Deployment course student but without the pressure of a team. The product ownership was given to program managers; these students assumed total responsibility for the product.

Technical tasks, including maintenance and release management tasks were assigned by a group of three students on 'The Engineering Change Order board' (ECO board). The ECO board evaluated work order proposals, approved or denied each proposal and assigned students to the accepted proposals. In essence, executive management of the courses was turned over to a group of senior students.

The team problems of the Construction course naturally disappeared since there were no teams. But there is no way to avoid the human problems; the team problems were replaced by tasking problems: e.g.

- Who is assigned to which task?
- Who gets to work with which task leader?
- What motivates someone to complete a task when there is no ownership?

The products improved markedly during the Deployment course while the students experienced the reality of working in a maintenance organization. The student learning outcomes were significant: the students experienced metric-based management, multiple version releases, configuration variants, and several other deployment issues.

The Deployment course was designed to have constant interaction with customers. We wanted the customers to use their deployed products, find defects, request enhancements and constantly interact with the students. Our biggest disappoint is that this didn't happen (for the reasons mentioned above). Without real customers, the Program Managers had to work in a vacuum and never came to fully understand their ownership role. The QA and testing processes suffered because there was no stakeholder who demanded an excellent product.

WHAT'S NEXT?

At the outset of the first year of these courses we anticipated problems identifying and defining suitable projects for industrial clients. Though the projects given the students suffered some problems, we have found that there are quite a large number of potential clients expressing enthusiastic interest in providing us with projects for the next year's sequence. Unless the need for software development expertise vanishes, we expect no problem providing the students with suitable challenges in the future.

However, we *do* need to make changes. The lack of strong commitment from the project providers, particularly in the last quarter of the sequence, diminished the learning experience. To ensure a sustained level of interest, we expect to ask the next year's providers for a full year commitment with a significant financial component. We also plan to work more closely with the project providers to increase the usefulness of the delivered systems.

Another important lesson is to look for projects in domains familiar to the students. The esoteric character of two projects distracted the students from the *software engineering* challenges. It is important that students learn how to communicate with non-engineers about unfamiliar domains. However, that learning objective is not well served by challenging students with exotic and technically difficult development domains.

Similarly, we will select projects amenable to tools and working environments familiar to the students. As with knowledge domains, the future professionals are expected to understand how to acquire skills with new tools as part of their professional development. However, if the tools required for a project require a steep learning curve then the skill acquisition process becomes a barrier to achieving more important learning objectives of the classes.

REFERENCES

Historical material

[1] Bauer, F L, "Software Engineering", *Information Processing* 71, 1972.

Material that discusses the need for trained computer software engineers

[2] Gibbs, W. W., "Software's Chronic Crisis", *Scientific American*, Vol. 271, No. 3, September 1994, p.86

[3] Office of Technology Policy, *America's New Deficit: The Shortage of Information Technology Workers*, U. S. Department of Commerce, 1998

[4] U.S. Department of Labor, *Occupational Outlook Handbook*, Bureau of Labor Statistics, 2000 (http://stats.bls.gov/ocohome.htm)

Material that discusses computer software engineering curricula topics

[5] Bagert, et.al., *Guidelines for Software Engineering Education*, Technical Report SEI-99-TR-032, Software Engineering Institute, October 1999 (www.sei.cmu.edu/publications/publications.html)

[6] Ford, G., *Education and Curricula in Computer Software Engineering*, Encyclopedia of Computer Software Engineering, Vol. 1, John Wiley and Sons, Inc., 1994

[7] Ford, G., *A Progress Report on Undergraduate Computer Software Engineering Education*, Technical Report SEI-94-TR-001, Software Engineering Institute, 1994 (www.sei.cmu.edu/publications/publications.html)

[8] Hiltburn, et. al, *A Software Engineering Body of Knowledge*, Technical Report SEI-99-TR-004, Software Engineering

Institute, April 1999
(www.sei.cmu.edu/publications/publications.html)

[9] Modesitt, Ken, "Software Engineering Program Survey Results", *Forum for Advancing Software Engineering Education*, Nov. 2000
(www.cs.ttu.edu/fase/reverse.htm)

[10] National Center for Education Statistics, *Classification of Instructional Programs – 2000*, U.S. Department of Education, 2000

Textbooks used in the courses

[11] Braude, E. J., *Software Engineering. An Object-Oriented Perspective*.John Wiley & Sons, 2001

[12] Brooks, F. P., *The Mythical Man-Month*, Addison-Wesley, 1995

[13] Gause, D. C., Weinberg, G. M., *Exploring Requirements*, Dorset House, 1989

[14] Grady, R. B., *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, 1992

[15] Hunt, A., Thomas, D., *The Pragmatics Programmer: From Journeyman to Master*. Addison-Wesley, 1999

[16] Jackson, M., *Software Requirements and Specifications*, Addison-Wesley, 1998