# Can Students Build Production-Quality Software?

**Gene Fisher**
**California Polytechnic State University**
**San Luis Obispo, CA 93407**

## Abstract

The question posed in the title of this paper has been asked in many forms. There have been thoughtful scholarly publications on the subject, and less than scholarly opinion pieces. This paper asks the question in the context of a year-long capstone course in software engineering, taught at Cal Poly University San Luis Obispo. Specifically, if product development is the overriding goal for such a course, can a team of senior-level software engineering students deliver and deploy a genuinely production-quality software product?

Unfortunately, the answer to this question in our case was "No". There are a number of reasons for the negative result, which will be examined in the paper. The examination will include consideration of whether it is reasonable to have product development as the primary focus of a university course, or if doing so sacrifices other important pedagogical goals.

## 1. Introduction

We have offered a year-long capstone course in software engineering since the 2000-01 academic year. The course was introduced at the same time as our degree major in software engineering, which we currently offer in addition to degrees in Computer Science and Computer Engineering.

Our capstone has been the subject of a number of previous reports, which have chronicled our progress[6,7,8,9]. Over its years of being offered, we have focused to varying degrees on a number of educational objectives. From the perspective of the students in the course, the objectives include:

1. apply the skills learned in introductory software engineering courses to a real-world software project
2. work with an external customer, on a project of specific interest to that customer
3. work in project teams of varying sizes, including in teams comprised of upper-class and lower-class students of software engineering
4. learn skills of project management
5. enhance technical skills of software development
6. deploy a working product of some form

The first objective is very common to engineering capstone classes across the disciplines. That is, students take what they have learned in lower-level courses and apply it in a setting beyond the classroom.

Our second objective is also a common feature of capstone courses, where students work with external customers. "External" means that the customers are beyond course instructors acting in the *role* of customers. For us, customers are often chosen from outside industrial partners[7]. In some cases, we have chosen partners from on our own campus[8], but still beyond the our own department.

The third objective for our capstone course, that of teamwork, is also nearly universal in a capstone experience. A somewhat unusual variant of our team structure has been to mix students from our introductory courses and capstone course in the same teams. This provides the upper-level capstone students the opportunity to manage the work of the lower-level students. This team structure was a feature of the initial offering of our capstone course[6]. It has not been repeated in many subsequent offerings, due primarily to class logistical difficulties.

The fourth objective of project management is again very common in a capstone class. In the cases where we have combined upper- and lower-division students, the capstone students managed their lower-division team members. In other cases, students would choose to focus on management primarily, or all students would assume rotating management duties.

The fifth objective of enhancing technical skills is important, but for us, as well as for others reporting in the literature, a secondary objective. In order to achieve the other course objectives, technical skills will invariably be improved. However, teaching strictly technical content is taken to be the objective of lower-level courses that precede the capstone.

Finally, the sixth objective, that of building a real software product, has always been an important part of our capstone. However, as with the fifth objective, the product itself has not typically been the primary focus of the course.

In our experience, fully achieving all six of the objectives has not been possible in any given year. We have therefore chosen to emphasize different objectives, depending on the instructional staff and the nature of the customers involved.

For the 2011-12 academic year, the focus was squarely on the product. In previous years, we had sometimes wondered if it would be advantageous to focus primarily on product delivery, to offer students a more real-world learning environment. The experience we gained in this effort, while not successful in its primary objective, will indeed help us continue to refine the course. It will also help us understand how to balance the different objectives to be achieved in a capstone.

## 2. Curriculum Structure of the Capstone Course

As noted in the introduction, ours is a year-long capstone experience. The curriculum is divided into the following three courses, each lasting for a ten-week quarter of instruction:

1. ***Requirements Engineering***
2. ***Software Construction***
3. ***Software Deployment***

These titles reflect a somewhat traditional sequence of software development, but in fact provide only general guidelines for the structure of the curriculum. Depending a faculty preferences in any given year, an agile development process may be used, where students iteratively analyze requirements, construct the software, and deploy it.

While software testing is not specifically listed in the course titles, it of course plays a key role. The faculty who teach the capstone have used methodologies ranging from test-first to test-last to points in between. In 2009, students conducted a controlled experiment to compare the effectiveness of different testing approaches[9].

## 3. Specific Structure of 2011-2012 Capstone Course

The course involved a project to build a course scheduling tool for our own campus. The need for such a tool had been well established, and a number of efforts had been undertaken in the past to produce one. Key features envisioned for the tool were the following:

- an easy-to-use database of instructor information, which includes course teaching preferences and teaching time preferences
- an easy-to-use database of a department's course offerings, including courses planned for particular quarters
- the ability to define department-specific scheduling constraints to guide the scheduling process
- a sophisticated scheduling algorithm that generates an optimized schedule, based on instructor preferences, planned course offerings, and departmental constraints
- the ability to fine tune a generated schedule, with automated checking to ensure schedule completeness and consistency

The tool is intended to be used at the department level, by the same people who normally perform department scheduling. The result of a scheduling session is in a form suitable for electronic submission to the campus scheduling database.

Achieving the features listed above is in fact an ambitious undertaking. An advantage we had in our efforts is a history of working on course scheduling projects. Specifically, versions of the scheduling tool had been assigned as two-quarter class projects in undergraduate software engineering courses for a number of years. In addition, there had been several senior projects that had refined key scheduling functionality, including the scheduling algorithm and schedule database management. The results of these past efforts provided a solid base on which to work. Using previous student work as a basis, we believed that a production-quality scheduling tool was an achievable goal.

What was significantly new about the project was to expand the focus from a small-scale department effort to a campus-wide effort. In late Spring 2011, a mailing was sent to department schedulers across campus. The mailing briefly explained the objectives of the scheduler project, and inquired about interest in project participation. Of the approximately 70 recipients of the mailing, nearly half responded with an interest in participating in the initial requirements gathering phase. A follow-on message was sent in the week before classes begin, to confirm continued interest in project participation. A total of 36 respondents responded affirmatively.

## 4. Related Work and the Definition of ’’Production-Quality’’

The goal of building production-quality student software has the been subject of a number of recent reports[1,2,3,4,5,10]. The curricula described in these works share most of the goals outlined above for our courses. Also common to these efforts and ours are the challenges faced when students endeavor to build production software. These challenges include:

1. finding suitable outside partners, from commercial or non-commercial organizations
2. logistical difficulties in collaborating with outside partners
3. clarifying deliverable expectations with the partners, including post-delivery work

The last of these points requires a clear definition of "production-quality" software, so that all concerned can be clear on the deliverables. Allen et al.[1] define the term "production programming" as "creating or modifying a software product to meet the needs of real customers". Others use a comparable definition.

While the Allen definition is useful, it does not refer specifically to the post-delivery needs of customers. In some cases it may be possible to deliver a stable and well-tested product to customers, with limited post-delivery support. However in today’s world of fast-changing software products, having a maintenance and evolution plan is an increasingly important part of a "production-quality" deliverable.

Post-delivery support can definitely be problematic for student projects. For example, Lange et al.[3] note that some of their customers have expected on-going "tech support" after student work on the project is completed. They indicate that such support must be clearly addressed in the initial project agreement, presumably indicating that it will not be available.

In several of the works cited above, there is discussion of different means to provide continuing product support. Such means include projects that continue across multiple years or making student products available to the open-source community. In any case, we believe that providing post-delivery support is a key part of truly "production-quality" software.

In our case, we did not adequately plan for post-delivery support, which is an important reason we consider our efforts not to have been successful. This topic is discussed further in the next section of the paper.

## 5. Results and Conclusions

As noted in the paper abstract, we did not achieve our primary objective of building a truly production-quality product. The specific reasons for our lack of success are the following:

1. The scope the project was overly ambitious, even given the substantial preparatory work that had come before.
2. The project became a fully ground-up effort, rather than being an upgrade or incremental addition to an existing system.

3. A specific technical decision to build a web-based application instead of a desktop application led to a number of delays in the project.
4. We were unable to secure a long-term maintenance agreement with any official organization on campus.
5. Students will be, and *need to be*, students.

The first reason for failure is of course extremely common in any engineering activity. For a group of students who cannot devote full time to a project, estimating project scope is a significant challenge. With a particularly well-organized and motivated team of students, producing a substantial amount of work may be possible. However in many if not most cases, the nature of academic work means that the scope of projects must be kept smaller than that envisioned for many useful software products. The lesson (re-)learned here is to be ever-mindful of defining student projects to have a suitably limited scope.

We had hoped to avoid the second reason for failure by using an existing base of operational code. However, we gave the students substantial latitude in choosing the project direction. Despite strong faculty recommendations to the contrary, it was the students' choice to re-develop from the ground up rather than using the existing code base. This aspect of failure reinforces the lesson from above. Namely that faculty need to be careful to control the scope of the project, and mindfully assert managerial control when necessary. This lesson is further reinforced by reports in the literature, including most of those cited in Section 4. Many of the successful efforts to build production quality products involve students adding incrementally to an existing project rather than building completely ground-up projects.

The third reason for failure is also common to software projects, that is, choosing the wrong technology. Understanding the development technology has long been recognized as a key factor in the success of a software project. While the students involved in our project had significant experience in developing desktop applications, only a few had web-based development experience. The steepness of the learning curve was far greater than anticipated. The lesson here is again one of asserting a reasonable amount of managerial control of the project. It may be OK to have a "sink or swim" policy for a student-run capstone project, and such a policy can provide a good learning experience for the students. However the "sink or swim" policy may well conflict with the goal to build production-quality software, which in our case it did.

The fourth reason for failure relates to the definition of "production-quality" software discussed in the preceding section of the paper. Even if the product had been suitable for deployment to the customers, it would have required some form of on-going development. We had hoped to make an agreement with a suitable campus organization to take over the project, but this agreement did not come to fruition. Understanding the importance of post-delivery development is one of the most important lessons we learned from this capstone project.

Last and not least, students must be allowed to be students. They are not full-time workers who bring pre-existing skills to the workplace. They must be given the opportunity to fail and learn from that failure. In this context, having students build production-quality software is always a challenge, and may not always be a realistic goal.

## Bibliography

[1] E. Allen, R. Cartwright, C. Reis, "Production Programming in the Classroom", Proceedings of the ACM SIGCSE Conference, Reno, Nevada, February 2003.

[2] S. Gorka, J. Miller, B. Howe, "Developing Realistic Capstone Projects in Conjunction with Industry", Proceedings of the ACM SIGITE Conference on Information Technology Education, Destin, Florida, USA, October 2007.

[3] D. Lange, R. Ferguson, P Leidig, "An Update on the Use of Community-Based Non-Profit Organizations in Capstone Projects", Proceedings of the ACM SIGCSE Conference on Innovation and Technology in Computer Science Education, Darmstadt, Germany, June 2011.

[4] M. Murray, "Implementing a Software Development Production Environment for Student Use: Advantages and Challenges", Journal of Computing Sciences in Colleges, December 2012.

[5] T. Nurkkala and S. Brandle, "Software Studio: Teaching Professional Software Engineering", Proceedings of the ACM SIGCSE Conference, Dalas, Texas, USA, March 2011.

[6] D. Stearns, S. Meldal, C. S. 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, Oslo, Norway, August 2000.

[7] D. Stearns, J. Dalbey, C. Turner, T. Kearns, "Report on a Capstone Project Involving a Hundred Students, for an Industrial Partner", Proceedings of the international Conference on Engineering Education, Valencia, Spain, July 2003.

[8] C. S. Turner, G. Fisher, D. Stearns, "Learning Software Engineering by Doing: Progress Report on a Capstone Sequence Involving Student Managed Teams", *Proceedings of the American Society for Engineering Education, Pacific Southwes Section*, Stockton, California, USA, April 2004.

[9] Vu, Frojd, Shenkel-Therolf, Janzen, "Evaluating Test-Driven Development in an Insustry-Sponsored Capstone Project", 6th International Conference on Information Technology: New Generations, Las Vegas, Nevada, USA, April 2009.

[10] H. Ziv and S Patil, "Capstone Project: From Software Engineering to Informatics", Proceedings of the IEEE Conference on Software Engineering Education and Training, Pittsburgh, Pennsylvania, USA, March 2010