Enhancing Software Project Management Courses With Industry Participation

Case Study in Cooperative Learning through Industrial Partnership

FECS'09 - The 2009 International Conference on Frontiers in Education: Computer Science and Computer Engineering

Dave Tahmoush
Computer Science Department
University of Maryland
College Park, USA
Tahmoush at cs.umd.edu

Sandro Fouché, Scott McMaster, James Purtilo
Computer Science Department
University of Maryland
College Park, USA
Purtilo at cs.umd.edu

***Abstract*—It is difficult to convince students that proper software engineering methodology is useful until they face a large project with serious deadlines based on legacy code. Without a realistic experience, students often do not understand why software engineering processes exist or are valuable. Without legacy code, the frustrations of software maintenance and upgrades are never truly imparted to students. By teaching students via a real software company experience with multi-semester projects managed by experienced students, teaching software engineering can become a rewarding experience for students with significant improvement in their future employability. The project management students gain significant insight into project management as well as a greater appreciation of software engineering when it is their project that they have to run. We describe how to teach very real lessons about software engineering, teamwork, and the realization that their code and client relationships will have be inherited by another class.**

Keywords- Cooperative; Co-op; Industry; Project; Education;

I. INTRODUCTION

The SEAM (Software Engineering at Maryland) cooperative brings together University of Maryland students taking CMSC435-Software Engineering with advanced students taking CMSC498S Software Project Management and with local community organizations to form an experiential learning experience that provides real software solutions for real world problems. Since its inception the program has included over 500 students and deployed approximately 20 projects. Student written software is being used in field situations by EMTs, Police, Fire and Rescue worker, scientists, and teachers.

SEAM is a curriculum and set of standard practices for Maryland's undergraduate software engineering class: CMSC435. CMSC 435 operates as a one semester capstone class for both Computer Science and Computer Engineering students. Consisting primarily of a large, semester long, group project, the class is intended to give graduating students practical experience before the move on to jobs in the industry. Initially conceived in the summer of 2001, the authors set out to revamp the existing "group project" class to address issues common in real-world software settings, including:

1. life-cycle development process
2. development tools
3. software support
4. project management

Divided into sections covering our goals, class structure and methodology, this paper provides details the authors' curriculum, best practices, and the resulting experiences over the last twelve academic semesters.

While this paper (and our curriculum) focuses on the semester project as a large part of student effort, our goal is to use the class assignments to drive an understanding of the importance of software engineering principles and practice. Everything from our goal, to class structure, and finally our methodology is designed to reinforce the issues presented in class lectures.

The industrial approach to teaching software engineering and project management has been applied previously [1, 2] and the multi-semester project [3, 5] with industrial partners [4] and with large projects [6]. This work has combined all of the lessons learned in previous work and enhanced it by simultaneously teaching a software project management course where students who have excelled at software engineering can manage a team of software engineering students in their project. This aspect of advanced student leadership and structured project management has significantly improved the experience of the software engineering student, the viability of the software projects, and the management of the course. The presence of student project managers gives a voice of reason, experience, and authority to the group project.

II. GOALS

First, our desire to revamp the existing group project class coincided with the matriculation of several graduate students

with extensive industry experience (including most of the authors). The common consensus was that undergraduates entering the industry for the first time had little, if any, practical experience, and required extensive introduction to production development issues and in some cases required retraining of habits they had developed in school.

CMSC 435 has always been envisioned as giving students a taste of large-scale development efforts. Unfortunately, the practical considerations of teaching a semester-long course limited the similarity of the coursework to real-world software engineering. This disconnect between the strictures of the classroom and the realities of real-world software was seen as responsible for the limited applicability of the classroom experience.

To close the gap, it would be necessary to bring the processes and mentality of the industrial development process into the University setting. Our interest was to change way students viewed the development process from a set of projects where success was measured in grades into a viewpoint that emphasized the importance of good software engineering process and practice.

A. Process

Our main goal is to emphasize the importance of good software engineering process to the success of real development efforts. Small scale student projects can often be completed in spite of a complete disdain for good practice; as a result students sometimes believe that software engineering is not necessary or does not apply to them. The authors, therefore, chose to implement relatively large-scale projects (10+ students), using real customers with real needs. The result is a demanding class (in time and effort), but one that makes good engineering process a necessity.

The authors agreed that the central issues in the development process–the ones that software engineering processes were designed to address–are those of communication. Communication between the customer, developers, testers, and others invested in the development process form the basis for most modern software engineering process. In the context of small group development (< 6 people) good software engineering practice might be limited or avoided altogether without seriously impacting the chance of success. As the size and scope of the development effort grows, so too does the need for good software engineering. The need for effective communication has become the driving factor in both the lecture and lab portions of CMSC 435. Students are presented with each phase of a life-cycle development process as a problem in communication. Specifically, students are asked to look at the development process as a series of questions:

- How does one illicit and understand Customer Needs?
- How does one convey understanding of the Customer Needs?
- How do developers coordinate their effort?
- How do you recognize the code works?
- How are faults conveyed to the programmers?
- How do customers recognized their needs are met?
- How do you refine your product?
- How do we recognize success?

Each one of these questions has as its answer a communication artifact of the software engineering process. As students begin to see the importance of communication as part of the development process, and their attempts to use ad-hoc communication methods are proven insufficient, the need for better process becomes more evident. Consequently, the need for good documentation–that reviled and often overlooked part of the development process–rises accordingly.

The process was improved through the inclusion of project management students who were familiar with the demanding schedule and the many potential points of failure. Their ability to motivate students into following process through examples of previous semester projects and through their detailed knowledge of the entire process leads to a greater buy-in from the software engineering students.

B. Mentality

Unsurprisingly the authors began the course re-design by considering their own changes over the course of their professional career, and one factor was immediately evident. As professional software engineers, we had a different mentality about our work. We took pride in the importance of high quality, efficient code, as well as manageable, repeatable (and let's face it sane) development process. Giving the students the same desire to see their classwork as more than just a means to an end would require a fundamental change in their attitudes. By providing students with motivation in the form of opportunities to excel and real tangible responsibilities we hoped to foster a more professional mentality in our students.

The inclusion of project management students who were familiar with the process did significantly improve the mentality of the software engineering students. Because there was a student who had faced similar distaste at having to document their work, but later found that it was exceedingly useful in creating a quality final project, the students were more open to the software engineering principles that they were learning.

C. Opportunities

In an industrial context engineers are motivated by variety of factors: pay, promotion, competition and even pride. Alternately students are usually provided with a single motivating factor for their work: grades. By requirement, grades are based on pre-determined criteria that are well exposed to students. Consequently, students often expend just sufficient effort to achieve their academic goals. While our CMSC 435 still relies on grades as a measure of academic progress, the new curriculum sought to illustrate to students the opportunities that exceeding their academic goals would engender. Notably, we sought to bring the same factors present in the industry into the classroom.

While pay is strictly absent within the classroom setting, but the students jointly own the code they produce in our class. While they are required to provide licensing rights to their customers, they are free to otherwise market, sell, and update their products.

Promotion is modeled within our curriculum in two ways. Successful students (those who achieve an "A" grade) are

eligible to return for a second semester of independent study working as the manager for future semesters (where possible we try to reassign people to their previous project). Additionally, since the projects completed as part of the SEAM project are used by actual customers, the students are encouraged to include the experience on their resumes. Anecdotally, students who participate whole-heartedly in the class report success in speaking about the experience during the interview process. The course has been unique in its ability to improve the employability of the students, giving them in-depth answers to typical employment interview questions like "Explain a project of your which fared poorly?" and "How many lines of code were in your largest project?"

While the students do not compete for the attention (or affection!) of the faculty or external customers, we foster friendly competition as part of the development process. The class is often divided into multiple "companies." The companies, while working on different projects for different customers, generally develop a mild rivalry to succeed. As one company moves ahead (completing requirements on time, meeting prototyping goals, etc), the remainder of the class is given clear indications of their progress in relation to their peers. The result is each group is motivated to keep up with the standards set by their classmates. One project management student noted, "challenging the team in the beginning to beat the other teams really helped to keep the team motivated and on-the-ball." Within teams, trivial and fun competitions are encouraged as part of team-building exercises (e.g. proposing company names, mascots, logos, bug finding competitions, etc.). While none of the competitions impact students' grades directly, the ensuing mental change (from meeting a pre-defined benchmark) to showing their capacity and prowess to peers, provides a subtle shift in the students' attitude to their work.

Pride sometimes operates at as factor in code development. Knowing that someone (or sometimes "everyone") uses your software – that you provide a necessary and useful function to others – can empower developers (or students) to refine and improve their code. Because SEAM integrates real customers with real needs into the classroom process, the students become part of a larger community (one including users as well as developers and instructors), the importance of the code they write becomes more evident. Additionally, peer recognition is often cited as a significant motivating factor for software developers. Throughout the course, students are encouraged to nominate their teammates to receive non-binding bonus credit for superb effort.

D. Responsibilities

For students to truly develop pride in their work, it is necessary to entrust them with actual responsibility. Too often students feel that class projects are merely "busy work", assignments that will never be put to use and are just reasons for teachers to assign grades. By introducing life-cycle support into our curriculum, the authors endeavored to make the students legitimately responsible for the upkeep and support of a product.

At the onset of each semester, students are assigned to product "company" – in most cases for a project worked on by a previous semester. As an introduction to their new role as developers in an existing project, an early assignment requires the students to review the project in terms of its software engineering principles. They get the previous project working with the help of install documentation written the previous semester, check for bugs, comment on the code quality and design, evaluate customer feedback, requirements, and design specifications, and provide written comments and a plan of work. As the semester progresses, students endeavor to support the existing version of the product as a mechanism to understand where improvements are necessary.

In due course, 435 students are assigned customer (re)assessment, requirements formulation, specification, prototyping, and implementation assignments. As a result is students are exposed to the entire product life-cycle, and in many cases develop a sense of accomplishment and pride in the product they create. Software is often published as an open-source project by the students.

A happy side-effect of the early emphasis on evaluation of existing project artifacts is the students develop a healthy respect for documentation. Understanding of an existing programming project can only be had from three sources: the developers themselves, the source code, or the documentation. We intentionally limit the availability of former developers for information as the students begin working on their semester projects. Students are forced to rely on the available documentation to understand the "how and why" of the existing work. Generally, the students bemoan the quality and quantity of their predecessors' documentation effort. In due course as each semester's students begin their own documentation efforts, they have a basis for recognizing deficiencies in their own documentation.

III. STRUCTURE

The SEAM initiative integrates with Maryland's standard undergraduate software engineering course, CMSC 435; given as a three credit class with two 75 minutes class sessions per week over an 12-week semester. Over the course of the semester the class session are divided roughly equally between lecture and lab sessions. During the early part of the semester most class are lectures, slowly giving way to more lab sessions during the latter part of the class.

The content of the lectures starts with an overview of processes and methodologies then as the semester progresses the lectures roughly coincide with the current phase of student product development – moving through the product life-cycle requirements, design, implementation, and testing. This approach allows students to immediately apply what they hear in the classroom to their work on the projects.

Early lab sessions include the introduction of tools and techniques by the teaching assistants and project managers, including: Subversion, UML, XML, AJAX, C#, and other topical tools. Later labs are used for project demonstrations with faculty observers.

Each section of CMSC 435 is usually approximately 35-50 students, which are broken into groups at two levels: small development teams of approximately four students and large product teams of approximately 3-6 development teams. The small development teams are meant to be highly coupled teams working on a single task set, but set at four so there would be difficulty voting on a correct course of action. The small size allows for frequent communication and close synchronization. Students are encouraged to select a team based more on

mutually compatible time-schedules than friendship or prior acquaintance. These small teams perform an initial small software project on an unfamiliar or challenging technology to get them working together well. Then the combination into the large product development team is then not as overwhelming.

The initial approach did not maintain the integrity of the small development teams in order to get all of the skills necessary for each team. However, the number of discontent students went down dramatically when the groups were maintained. The integration into a large group appeared to be socially isolating without the small group structure. When students were given the option of breaking their groups, only one students out of three hundred opted out of his group, and then only for a project that he was very excited about.

Project managers were allowed to present their projects to their prospective students, and the students ranked their projects. This often matched well with the recruiting desires of the project managers, who would look for groups with specific necessary skills. Project managers would assemble their teams with a draft approach, picking groups that had the technical skills needed for their project. This led to occasional problems, like a team that had no database specialist because the project did not require it until the requirements changed.

IV. METHODOLOGY

A. Communication

Software development is in large part a collaborative activity. Therefore, SEAM emphasizes group dynamics throughout all phases of the experience. Project team sizes are usually ten to twenty students, with one project management student assigned to be a "project manager". The project management students are prohibited from coding or working on documentation to prevent them from allowing themselves to be distracted from their management duties. They primarily serve in a management, mentoring, and advising capacity. One project management student noted, "failing so miserably to properly communicate and manage this project is probably the most important thing I've learned in four years of college." This student later went on to manage a new project successfully after having his project cancelled by the client. The project management students eventually realize the importance of strong communication within their projects. One student noted, "Micromanaging talent, recognizing strengths and weakness, and properly meeting a timeline and budget are impossible or irrelevant without a strong means and enforcement of communication at all levels."

The larger groups had more complicated communication problems. The use of documents and interfaces become more important, as one student noted, "if we could start back again from the beginning, (I would have) interfaces play a bigger role, as we're communicating between a bunch of different teams and making frozen interfaces would have helped immensely about a week ago." Having the students work on these large projects helps them to focus on the communication difficulties. One student noted, "wonder whether organizing the project into those three groups was the best idea. I know that we certainly needed a leader for each of those sections to make designs. But maybe we should have assigned people features, or portions of the project, to work on instead of telling them to assist a leader. That way they would be completely responsible for their portion from start to finish and would not have to rely on other people (or use them as an excuse). This seems like it might cause coordination problems but that is what the leaders are for." Whether the student was correct or not, he did start thinking about the organization and the communication of responsibilities.

Students did find ways to overcome the communication difficulties inherent in any large project, from using systems for testing and tracking bugs to creating useful meetings. One team noted, "Some of our group met to draw the interface out on a whiteboard. This worked well because we were able to quickly bounce ideas off of each other." The value of effective communication and interfaces are also realized as one student noted, " Our front-end and back-end have been developed in a vacuum and there was little communication between the two (except the interface). It worked with no modification."

B. Good Software Engineering Process

Though testing is a significant part of software engineering, it can be difficult to motivate proper testing of student projects. Most senior students have become proficient at unit testing, but remain ignorant of the intricacies and difficulties of integration testing and system testing. Since students often equate more features to a better product, they rush to incorporate more minor features at the expense of adequate testing. Students are also inherently overconfident in their product, and need to be shown the lack of quality in their code in experience as well as in theory.

Students usually discover the value of testing only after it is almost too late. One student noted, "We ran our JUnit tests after every fix to make sure nothing broke. The problem was our JUnit tests were not comprehensive. Since we were in a rush, we didn't test every feature. This meant that in some cases, our bug fixes create other bugs. We really should have written complete JUnit tests to avoid this problem." Though the students were taught about testing coverage and tried to implement regression testing, they only truly appreciated the value of good testing coverage once they discovered that new bugs would be created if they did not have adequate test coverage. Another student noted, "Testing was interesting. I found some bugs in my program that I had not found in testing while programming." Students also discovered that bugs could be found in many places, "some coming from old code, some from our code, and some from the image tool we imported into our code."

An effective method for exposing the lack of quality in student projects is the use of cross-testing between different projects. After the final project is finished, students switch projects and evaluate the quality of the other projects. Final projects include a list of known bugs to facilitate continued testing as well as maintenance. Cross-testing enables the students to practice testing on a different project as well as receive a report on bugs that they were unaware of in their final projects. This was effective in teaching students about the level of quality of their work.

Typical student responses to the cross-testing of their projects range from frustration to hasty explanations. Student projects often include packages available as free-ware or publicly available code-bases, which are often considered to be bug-free but produce multiple significant bugs under cross-testing. Often bugs are found in the overlooked areas of security and networking.

C. Customer Oriented

SEAM's industry partners are busy individuals, and being respectful of their time is essential to maintain an ongoing relationship from semester to semester. To ensure that students do not drive customers away, they are given guidelines as to the nature and frequency of customer contacts. They are coached to batch and prepare a list of questions for the customer well in advance of each meeting, to hold each meeting to a fixed length of time, and to designate one or two team members to serve as customer liaisons.

Communication with the client is stressed, as well as frequent presentations of current progress. One student noted "the most important thing when presenting is (the software's) appearance. A buggy UI makes for a much worse reaction than if other components have far worse bugs." Meetings with clients help students to ensure that their software is in-line with the expectations of their clients and that they have successfully captured the requirements of their clients. Demonstrations have caught many mistakes earlier in the process, and one student described them as "useful. (The Client) basically tore many aspects of (the project) apart." This interaction with a client who is genuinely interested in having a working project is one of the improvements in the learning experience.

D. Customer Oriented

Two essential aspects of building a software "product" that is often overlooked in traditional educational projects is the need to include installation instructions and a setup program in the final delivery, and to provide support. In our experience, SEAM students often struggle with the installation requirements. Installation is usually first addressed by less-experienced engineers toward the end of the product development life-cycle. Consequently, quality may suffer due to time pressures. Also, this tends to be students' first significant experience with installation technologies, and thus a learning curve is involved. One student noted, "It was interesting to deploy a piece of software that was truly standalone with an installer and GUI and all that. I'd never done that before."

Product teams are informed in advance of the target deployment environment and are responsible for ensuring that their products can be installed there by a novice user. Students are informed of the general timeframe during which installation and acceptance testing will occur and must supply "on-call" support. If installation fails for any reason during grading, students are initially given no grade and a fixed time period in which to resolve any issues with the graders.

While students are allowed to choose any software development life-cycle for their project, the pragmatics of a semester-long course usually lead to a waterfall process: Product teams have interim deliverable dates for requirements documents, design specifications, and other typical artifacts, and a semester is generally too short to execute multiple spirals in any event.

It is difficult to expose students to the difficulties of maintaining and upgrading an older code base and train them to work well with inherited code in a standard course environment. The use of clients with established code bases and structural needs, as well as continuing and upgrading earlier projects, provides students with what is often their first exposure to the difficulties of inherited projects. Inherited projects are initially tested and evaluated by the students on the basis of software engineering principles which forms a basis for redesign or reengineering a project, as well as familiarizing students with the project.

V. Software Project Management

During our first semester our management structure was very flat: a single professor, acting as the CEO to all companies. A single TA, acting as VP of Technology to all companies, and group leaders elected from within each group. The feedback at the end of the semester was surprising. Several students recognized the need for people with more power than the student-elected group leaders, but at a more hands-on level than was provided by the high-level management from the instructors. They had recognized the need for middle-management.

In subsequent semesters we added more management by bringing students in for a second semester class on software project management. Initially given as an independent study course, and only available as an option to students who achieve an "A" grade in CMSC 435, CMSC 498 students have become an indispensable part of the SEAM initiative. These returning students, under the guidance of the instructors, act as managers to the current semesters 435 companies. 498 students are required to refrain from doing any development and are restricted to only management and guidance activities. Their grade is based on their projects level of success, the feedback from their managed students, participation in specific discussion sessions with the instructional staff, and an end-of-semester report on their insights into software engineering process in relation to their experiences managing a software team.

Uniformly, 498 students show enhanced understanding and knowledge of the development process and how software engineering process addresses the problems inherent in large-scale development. Anecdotally, the 498 students have raved about the effectiveness of discussing their experience in managing the software process and the lessons learned while interviewing for jobs. Several students received jobs in program management instead of the more traditional software development after completing this course. While all of them feel that the (independent study) class is demanding, they also see it as a critical part in their post-academic success. One student noted that, "(the class) provided real world experience without the cost of getting fired from the many mistakes I made."

VI. Conclusions

Anecdotally we know our students see greater demand among employers, who tell us they feel more confident making hires from the class (rather than stick with a practice of only hiring employees who have been "trained by the street" for a few years in industry.) These students have already had their first practical experiences, but in a controlled environment. We also see the first generations of students through this class coming back to both tithe into the classroom setting (providing guest lectures and helping freshen up the instruction of particular tools) and to seek fresh hires for their own companies. We find new companies with which we've had no prior contact specifically seeking access to these students, and hire them directly out of the class.

ACKNOWLEDGMENT

This work was partly funded by the University of Maryland Graduate Research Board.

REFERENCES

[1] M. Bernstein, K. M. FitzGerald, J. P. Macdonell, and A. I. Concepcion. Algorithma project: the ten-week mock software company. In SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education, pages 142–146, New York, NY, USA, 2005. ACM Press.

[2] J. M. Clifton. An industry approach to the software engineering course. In SIGCSE '91: Proceedings of the twenty-second SIGCSE technical symposium on Computer science education, pages 296–299, New York, NY, USA, 1991. ACM Press.

[3] D. Gotterbarn and R. Riser. Real-world software engineering: A spiral approach to a project-oriented course. In CSEE, pages 119–150, 1994.

[4] J. V. Harrison. Enhancing software development project courses via industry participation. In CSEET, pages 192–203, Washington, D.C., USA, 1997. IEEE Press.

[5] D. H. Hutchens and E. E. Katz. Using iterative enhancement in undergraduate software engineering courses. In SIGCSE '96: Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, pages 266–270, New York, NY, USA, 1996. ACM Press.

[6] D. E. Wilkins and P. B. Lawhead. Evaluating individuals in team projects. In SIGCSE '00: Proceedings of the thirty-first SIGCSE technical symposium on Computer science education, pages 172–175, New York, NY, USA, 2000. ACM Press.