An Industry-Academia Team-Teaching Case Study for Software Engineering Capstone Courses

Amalia Rusu and Mike Swenson
Fairfield University, arusu@mail.fairfield.edu, mikswenson@msn.com

Abstract - Exposing software engineering students to newest industry practices and latest research and theories allows them to acquire and maintain the technical skills necessary to continually adjust to the rapid changes that occur in technology. Close interaction with industry members helps the university and engineering programs identify real-world problems and their solutions and incorporate them into the curriculum. Software engineering capstone projects require the development of major software products and are usually either industry-generated or research-based. They are either assigned for the overall supervision of one instructor or each project is supervised by different instructors independently. In this paper, the authors present a case study in which the software engineering capstone projects have been team-coordinated by two instructors: one full-time faculty and one part-time faculty (full-time industry practitioner), thus combining two complementary sets of skills towards the mentoring of the software engineering students. Two capstone projects (one industry-generated and the other research-based) are also presented with the contribution of each instructor described.

Index Terms – Capstone projects, Industry-academia partnership, Software engineering education, Undergraduate and graduate.

INTRODUCTION

The Software Engineering curriculum usually provides room for mandatory capstone projects of one or two semester length. Unlike the majority of undergraduate or graduate software engineering courses which are taught by presenting the concepts and methodologies and by assigning various reading, small assignments, and individual projects, in the capstone courses, students work in teams, perform a technical study, and design and develop a software management or development tool. Various approaches ([1]-[2]) simulate the industry working environment by having the entire class work on a single project with groups of students assigned to particular tasks. However, while this approach may resemble the industry practice in some medium-size companies, the fact that each group of students focuses and practices only on a specific set of skills may restrict students’ opportunities after graduation. Other approaches give the students the opportunity to exercise their software engineering skills on projects that closely resemble the industry environment by allowing students to practice all software engineering phases and instead of working on generic problems rather focus on industry problems which are more specific, focused, time bounded, and require team work ([3]-[7]). The disadvantage of this approach is that many times students cannot succeed in producing a final workable product due to the limited time available. However, it has the potential to be successful if there is a two or three semester course sequence.

Nevertheless, increasing the involvement of industry in academia not only will prevent the skills shortages in the future but also increase the pool of qualified technical people with practical work experience for certain jobs. For instance, software engineer occupations are projected to grow the fastest over the next decade. However, in order to acquire and maintain the technical skills, computer software engineers must continually adjust to the rapid changes that occur in technology. Close interaction with industry members helps the university and engineering programs identify real-world problems and their solutions and incorporate them into the curriculum.

Academically-oriented instructors (usually full-time faculty) and industry-oriented instructors (usually adjunct-faculty who are also full-time industry practitioners) bring slightly different viewpoints to the coordination of senior and capstone projects. Because of their academic duties, full-time faculties do not have the time to be exposed to newest industry practices by actively participating in industry projects. On the other hand, industry practitioners do not have the time to stay up-to-date with latest research and theories.

In this context, we present an industry-academia team-teaching case study for a two-semester software engineering capstone course team-coordinated by two instructors with complementary sets of skills.

OUR APPROACH

Senior project or capstone project courses are typically upper-level undergraduate or graduate courses in the Software Engineering curriculum and best suited for team-coordinated instruction. Generally, the students enrolled in these courses already have a strong programming background. However, students usually have not developed appreciation for careful planning and have not yet produced the extensive and comprehensive documentation related to large projects. The majority of the time their assigned projects had clear descriptions, with clear grading schemes,
leaving little room for questions and interpretation. On the other hand their experience working in teams may also have been limited.

The root of our approach is the belief that the best way of learning is by doing and the best way of doing is by actively participating in larger industry-generated or research-oriented projects and experiencing various models of the software life cycle.

At the beginning of the two-capstone course sequence, appropriate software engineering projects have been carefully selected by the instructors that can be successfully completed within two semesters and at the same time having industry marketability potential. Both instructors have been involved in students’ supervision throughout the two semester sequence of capstone projects.

The next step in the process is to divide the class into teams and assign each student to one of the projects. All teams will then go through the entire software lifecycle by working separately on their own project. The class is divided into teams of four or five students. Other approaches ([5]) have used teams of more students to achieve more in a shorter period. However, dividing the class into such larger teams does not always work in the academia environment since students have different course load, busy schedules, and become very hard if not impossible to coordinate them for outside class team meetings. Then the instructors’ responsibility is to assemble teams of complementary skills to mimic those found in software development businesses in industry.

The instructor acts as the project manager for the projects. The instructor also assists the teams’ development process and provides help and inputs to students without influencing much or limiting their creativity. In this way the students have full control over the development of the software. However, after completion of each phase, each team receives a critical analysis of their deliverables.

The two projects used as the basis for our case study are called iExercise and MatchIT and described in the next section.

PROJECTS

I. iExercise

iExercise is an industry-generated application driven by the need to provide users with more flexibility in the type of diet and exercise programs in which their progress can be tracked. The fitness industry is a multi-billion dollar industry that ranges from exercise equipment to supplements to gyms/spas to workout apparel. In recent years the explosion of internet activity has led to fitness enthusiasts meeting online to discuss progress, problems or ideas on discussion forums and email lists. Some have even used websites to record their progress and share it with others. A number of websites exist that are database driven and support this functionality. Most websites are specific to a particular niche, such as weightlifting, cycling, running, dieting, etc. What does not exist is an intelligent exercise tracking system that is flexible enough to serve the needs of a varied clientele and provide functionality and analysis specific to each user. iExercise is the system that bridges this gap and uses the latest technology available to give any individual the tools to track, analyze, set goals, share their experiences and improve themselves. The key to reaching a fitness goal is having a good understanding of where one is at and what is needed to reach that goal. iExercise gives the user the tools to maximize their efforts.

In particular, iExercise is a web based application that one of Fairfield University’s software engineering capstone teams developed in C# using ASP.NET 2.0, hosted on a Windows 2003 Server with SQL Server 2005 as the database backend (Figure 1). iExercise allows the users to customize their fitness profile, with the exercise and diet modules they select. Within the selected modules, users will be able to enter exercises performed and be able to track their progress over time. iExercise also provides a set of tools to users that will allow them to calculate and analyze their exercise data, track notes about what they are doing, and share information and results with other users in the community via a bulletin board. In addition, iExercise is an improvement over other sites with its simple user interface and modular design which allows for easy addition of other fitness modules and future enhancement.

![FIGURE 1 IEExercise GUI.](image)

II. MatchIT

MatchIT is a research-oriented project that proposes to develop a fully functional multi-modal biometrics identification system. Companies have certain areas in their facilities that contain sensitive employee information, valuable intellectual properties or physical objects. Security for these areas is necessary to prevent unauthorized access. ID cards, user identifications, pins, and passwords can be easily acquired or stolen. A Fairfield University software engineering capstone team has developed a fully functional person authentication system, called MatchIt, that combines the face and fingerprint recognition to get the top matches
and display data to the user to confirm the identity or deny access. MatchIT addresses the security issues by exploring the use of biometrics to verify an individual’s identity. The students have explored facial and fingerprints recognition technology by building an application that matches individuals by various characteristics. They have used eigenface vectors in the identification process whereas for fingerprint matching, they have used minutiae points characteristics. MatchIT has the ability to return the top closest matches and pass them to the user for validation purposes. The team has used various biometrics matchers for higher security and combined multiple classifiers to improve the recognition accuracy.

MatchIT has been developed in Java using Java Runtime Environment (JRE) 1.5 and Hibernate to connect to the Oracle database. A multi-modal biometrics search algorithm has been used, which starts by selecting the face and fingerprint of an individual to be matched to the database (Figure 2). The corresponding face and fingerprint input files are loaded into an area that has various browsing buttons. The algorithm ensures that the face and fingerprint images belong to the same person. It will receive a matching pair of face and fingerprint images from a frontend application that will capture both images at the same time and name the pair of images with the same filename prefix. If only one filename is provided for the biometric search, then MatchIT has the ability to search on individual biometrics, as well as multi biometrics search otherwise. There is a default threshold value set up for making the cut for a list of images that will be displayed but the user also can change the threshold to any value within a range. Each type of biometrics has its strengths and weaknesses. However, multi biometrics combines different biometrics to get a more accurate result than by using a single biometric approach.

![Figure 2: MatchIT GUI](image)

**Differing Approaches to Capstone Projects**

I. First Semester

First semester has been taught by a part-time faculty who is also a full-time industry practitioner, with the other instructor monitoring the class. The assumptions for the projects’ first phase were:

- Students learn in a waterfall approach, initial classes lead to successive ones, learning analysis and requirements gathering before design, implementation and testing.
- Budgets for time and resources are finite or restrictive in the classroom setting as opposed to scheduling and staffing flexibility in business.
- In industry, development teams, especially when developing a new system, apply newer techniques and tools where appropriate. The learning process never ends.

**Motivation:** Problems of completion. Using a waterfall approach leads to the problem of completion. If students tackle more complex problems, there usually is not enough time to complete a full software development life cycle. On the other side of the complexity scale, students choosing rather simplistic projects do not get to understand more real world software engineering problems and the applications tend to the trivial. An alternative approach is called for.

**Background:** Industry prefers seeing concrete running deliverables earlier, rather than later in a project. This provided critical feedback earlier in the process. A multi generational project plan is often used for large, multi-year projects in industry. Time boxes are used to help show measurable progress towards a goal. Forward thinking development teams are driven by several factors and when developing a new system, apply newer techniques and tools where appropriate. The learning process never ends.

**Challenges:** In the context of the capstone projects, there are several challenges (see motivation above) to actual completion of the course. There are also several educational challenges. While it is assumed that the students all have completed the prerequisite coursework, this does not mean that all students have taken the same coursework. There are a variety of concentrations in the program and not all students take the same courses. This does produce gaps in skill sets. There is also the challenge of integration, the reason for capstone, of the student’s learning. They may know what to do (steps to take, artifacts to produce) but not necessarily why. They also must use their own artifacts as inputs to next steps (they eat their own dog food). Projects that begin to fall behind have four options in industry, extend the delivery date, add resources, remove features and skip or short change particular steps or aspects of a project. Capstone students do not have the first two options available and must choose from the latter two.

**The Approach:** The adopted approach to the first semester capstone course has changed over the past 4 years, evolving from a traditional waterfall approach to a more pragmatic iterative approach. This approach for capstone projects has been to use a multi-generation project plan to
scope the full project concept then use a time box approach for actual project deliverables. A multi-generational project plan is often used for large, multi-year projects in industry.

At inception, the teams decide the project, envisions what the 3-5 year end goal of the idealized project would be, this is termed version 3. Then, the team works on the minimum set of requirements that provide user value, this is termed version 1, version 2 is generally determined at a later date but would consist of a subset of version 3 features. It is assumed that version 1 can be completed within, sometimes well within, the two semesters allocated so version 2 gets features that can be completed by the end of the program.

The time boxes are a way to show progress throughout the term. Generally they are 3-5 weeks in duration and provide significant accomplishments. Goals are determined by the team and agreed to by the instructor. Students learn to produce accurate estimates of their work. As a rule, there are also running software components delivered. Each time box work item will contain tasks that contribute to the current objectives (time box goals) as well as some research or work related to future deliverables. This leap frog approach is often used for tasks that may span multiple time boxes.

Results: Having taught capstone using several approaches, traditional waterfall, software studio approach, and now the multi-generational-time box, this latter approach allows the students to complete at least one complete cycle in software development. It has also allowed for teams to make tradeoffs in development knowing there are later times in a projects lifespan to go back and do it “right”. We have also had teams pick up where previous capstone teams have left off, this has had the effect of teaching students not just what project artifacts to leave behind but learn directly how these should be organized and to what level they should be created.

Lessons Learned: Students need to be engaged to succeed, and not all students are engaged by the same aspects of a project.

II. Second Semester

Second semester has been taught by a full-time faculty, with the other instructor monitoring the class. The assumptions for the projects’ second phase were:

- The requirements and design for the first phase of the projects have been implemented by the end of the first semester as a first software version.

Motivation: With new requirements for the second semester, we have started the Rapid Prototype model for the software life cycle. In the absence of an outside customer, this model has been proven to work better in practice since the requirements are unlikely to change. Moreover, providing a user-friendly graphical user interface is a requirement for the projects, therefore this model has been considered and adopted. Some of the benefits of Rapid Prototyping are that it helps in detecting the missing functionality and identifying confusing services earlier since it starts with those requirements which are poorly understood resulting in solidification of requirements, also it has planned project stages, and allows for project management techniques, which is better than chaos. Moreover, a prototype is available early in the process (i.e., at the beginning of second semester, making use of version one of the software developed in the previous semester) and used to construct the specification document which lessens the need to repair the design during the implementation.

Background: Once the semester starts, the instructor presents a timeline for all the deliverables for the class and assists students by providing summarized lecture notes on all aspects of modern software engineering to maximize the students’ learning curve as well as minimize their time spent on the assignments that might require reviewing materials taught in previous software engineering design classes. Since most software projects fail due to lack of time to complete, the instructor must provide enough details on the documents that need to be delivered including sample documents for each software lifecycle phase. This greatly helps the students to structure their documents in a professional manner.

Challenges: Unlike the first semester of the projects when the students started developing software systems from scratch, in the second semester we require to extend and modify the existing systems, which usually adds complexity to the integration task. Also learning objectives could vary throughout the two semester sequence of capstone courses and that become a challenge for students to rapidly adapt to the new goals. There may be also differences between team experiences due to the nature of the projects. For instance, one team may advance easier to the implementation phase whereas the other one can still struggle in learning about new research technologies and algorithms that can be applied.

The approach: In the second semester the students used a Rapid Prototyping lifecycle model and experienced all the phases of the software development process one more time as they produced a fully operational software component. It is important for the students to play an active role when participating in a software engineering project. In our approach, each team is expected to:

- Produce a detailed project proposal including new requirements and any work left untouched from previous semester.
- Submit weekly team logs to allow the project manager (i.e., the instructor) to follow and comment their progress.
- Perform requirements analysis and specification for the new requirements.
- Design new modules and change the existent architecture of the system accordingly.
• Perform structural and specification-based testing.
• Produce supporting documentation.
• Prepare a professional customer-oriented presentation (i.e., poster, presentation, user manual, demo of the final software implementation).

In addition, the students learn how to make presentations to both technical and non-technical audiences, write coherent well-reasoned reports, and assess the implications of the technology they create. Since the students have different backgrounds and skills, not all of them coders, there should always be various tasks, central to the project that team members can successfully complete. Various amount of text is written and edited in each portion of the project and significant testing is required by the final phase.

**Results:** The excitement of working on projects that matter in industry and research increased the level of motivation for the students. At the beginning they learned the significance of choosing the appropriate lifecycle model for the job. Using the rapid prototyping model allowed the students to efficiently use the time and focus on the deliverables for each phase individually (requirements, design, implementation, and testing). Project phase planned intervals varied from 2 weeks (for testing) to 3 or 4 weeks for the others. In the design phase of the software engineering process a greater appreciation for the requirements specification was gained, and consequently in the implementation phase the students found it easier to code having a detailed software design document for reference. Their previous experiences with testing had been limited to boundary testing, and formal methods had never been emphasized. Testing phase exposed the commitment necessary to develop quality software. The students needed to provide the following sequence of deliverables by the end of the second semester, including the work from the first semester as well: weekly team logs, project proposal, software requirements specification document (including the prototype), detailed design document (including the integration test plan), project code, implementation report, user manual, testing document, a midterm paper, and final presentation.

**Lesson Learned:** Learning to accept new ideas and criticism was not easy but we knew that we were ultimately working towards a better product. The students also learned the importance of researching details before embarking in a new technical or research domain. Well-structured software development takes time, and the benefits are not immediately apparent. However, it benefits the projects in the long-term.

**CONCLUSIONS**

In this paper we have presented a case study in which a two semester sequence of software engineering capstone projects have been team-taught by two instructors, one full-time faculty and the other part-time faculty and full-time industry practitioner. Their combined complementary sets of skills have been used towards the mentoring of the software engineering students and greatly improved students’ performance in the projects.

Our approach addresses the forthcoming challenges students will face in industry and ignites their entrepreneurial spirit. Students were challenged with deadlines and responsibilities to perform at a professional level. They discovered that time management, careful planning and monitoring, and timely, and thorough coordination are crucial for software engineers. All students benefited from this maturing experience and, with their broadened perspective and keener insights, are better equipped to deal with any phase of software development, as well as with basic aspects of project management. Two concurrent paths needed to be used to successfully complete these complex projects. One is the project itself, the other is to study relevant software engineering topics in textbooks such as [8]-[12], and researching new technologies and research topics.

By removing the anxiety of having to have a project perfectly scheduled over two semesters, students can learn from each time box or phase how to more accurately estimate their work efforts. Students can learn new skills in the context of the project and apply them immediately. We have found this approach leaves us as instructors more opportunities to teach the value of continuous learning in the context of a project. Successful software engineers need to be learning and adopting new techniques and tools throughout their careers. This is not limited to their technical skills but also learning different business domains and preparing for different roles.

On the other hand, the documentation prepared throughout the process ensures the correspondence between the projects’ documentation and students’ implementation, allowing for smoother transition of projects’ versions from one team to another for future enhancements. Nevertheless, the techniques described for the first semester as well as for the second one can be adopted and used in upgrading existing systems as well as for new systems.

In addition to the exposure to the variety of software engineering methodologies and good industry practices as well as various research topics, the complexity of the projects and advice from an industry-academia team of instructors give students flexibility in assuming a more active role to development of the software systems after graduation.

**ACKNOWLEDGMENT**

We thank team iExercise and MatchIT for their hard work and commitment to the capstone projects.

**REFERENCES**


©2008 IEEE


