Experiences Using Student Project to Create University Business Applications

Harry Koehnemann and Barbara D. Gannod
Arizona State University East

Abstract

Engineers (both hardware and software) are continually developing and testing processes to create systems "better, faster, and cheaper." A variety of software processes have been developed by the software enginnering community toward this end. Among these processes are eXtreme Programming (XP), Crystal, Feature Driven Development, and the Rational Unified Process (RUP).[1] To be attractive to potential employers, students in computing fields that intend to become software developers should be exposed to and, ideally, have practical experience with modern software processes. This paper describes experiences in a software capstone course which teaches students the activities associated with two popular industry processes: XP and RUP. In particular, the course uses student projects to create applications used within the university. The unique aspect of the course that differentiates it from other software engineering capstone experiences is the emphasis on agile processes (primarily XP) and the use of software development tools (e.g. configuration management, automated testing, modeling) commonly used in industry.

Four university projects have been created to date. The first is a web application that supports an NSF funded curriculum development project. The second is a channel supported by the uPortal portal system that automates the department's graduate admissions process and is deployed on the university's IT portal system. The third is an application that integrates a Course Management System, Blackboard, with an outcomes based assessment tool, True Outcomes, to automate the importing of student information to better measure outcomes for ABET accreditation. The fourth project is a linguistics analysis tool that finds word usage patterns in media articles.

1 Introduction

One of the program characteristics that ABET expects in engineering and technology programs that it accredits is the inclusion of some type of experience that allows students to integrate diverse elements of their education.[2] Most programs implement this integrating experience as a capstone course (or course sequence). A capstone experience is typically a culminating experience in the specific major that allows students to creatively apply principles and methods acquired throughout their education to a significant project having a professional focus. Ideally, the capstone experience should be as close an approximation to the “real world” as is possible in an academic environment.

This paper describes experiences in a “Software Factory” class that is the culminating experience for majors in Computing Studies at Arizona State University (ASU) East. In order to provide experiences that prepare students for careers as software engineers, we believe the following aspects of the course are vital:

- Students should work on real projects with real customers.
- Students should follow real processes to develop and/or maintain software artifacts.

Traditional academic software projects, even team projects, often do not provide a realistic software development experience. Typically, their results are not exercised by real customers, and the resulting code is never enhanced or augmented to meet new and changing requirements. Students therefore do not experience common development issues such as the consequences of poor design decisions, poor quality procedures, bad requirements management, and mismanaged customer expectations.

Further complicating the problem is the fact that most computing curriculums have not kept pace with software practices. Most Software Engineering curriculums teach high-level concepts around a traditional waterfall lifecycle approach to development. New methods and techniques emerging in industry, such as eXtreme Programming (XP) and Rational Unified Process (RUP) and their guiding philosophies have not yet made the transition into most curriculums.

To provide a more realistic experience, the “Software Factory” course (CET415), a capstone course on applied software process, focuses on the use of projects with real customers and the adherence to real software processes during the development of the project. Customers have been taken from the university and local community, and the course allows student to apply XP and RUP to develop and deploy their software solutions. The innovative and unique aspect of the course is the use of agile software processes rather than traditional software engineering methodologies.

The remainder of this paper is organized as follows. Section 2 gives a brief survey of approaches to capstone courses and also provides background on the software processes used by the “Software Factory” course described in this paper. Section 3 discusses the objectives and course organization for the software capstone course. Section 4 describes the projects and experiences of the capstone course, and Section 5 provides a discussion of experiences and lessons learned through the offerings of the course. Section 6 concludes the paper.

2 Background

2.1 Capstone Experiences

Some flavor of capstone experience is found in most computing curriculums. Naturally, the nature of the experiences varies widely. A comprehensive survey of capstone courses is beyond the scope of this paper; however, a representative set of capstone course descriptions is provided in this section.

In general, capstone projects may be "canned" projects invented by a course instructor or they may be "real" projects with customers taken from the university or local community. Additionally, projects may be carried out by a single student or by a team of students. In a brief survey of capstone courses at many universities, we have found that using real projects is commonplace. In general, courses emphasize (among other things) requirements analysis and customer interaction, making real customers an important component of the experience. The size of the development team has greater variance. For example, Beasley suggests that a single student should carry out a capstone project.[3] However, it appears that a small development team is more common. Many courses emphasize project management and teamwork, making teams a vital component of the course. For example, a representative capstone course at Michigan State University utilizes teams to develop software for customers from the local industry such as Motorola and the Ford Motor Company.[4]

Regardless of whether capstone projects were carried out by single students or teams, or whether or not they are developed for real customers, a common thread is the predictive nature of the development process for the projects. For example, various artifact such as a software requirements specification document, a design document, a test plan, a software release, and a user's manual are given specific due dates. For an example, see the capstone course offered at Southwestern University.[5] This common organization follows a very traditional ("waterfall"-like) approach to software development. In terms of software processes, described in the next section, a one or two semester approximation of a heavyweight process is used almost exclusively in software engineering capstone courses.

2.2 Software Processes

Engineering disciplines recognize that in order to build quality *products*, quality *processes* must be followed. This was the motivation for the Capability Maturity Model (CMM) defined by the Software Engineering Institute (SEI).[6] In general, a software process includes both the methodologies for developing software and the plan for software development. Software processes can be categorized as heavyweight or lightweight processes.

Heavyweight processes and RUP

A *heavy* (or *heavyweight*) process is typically used to suggest a software process in which a relatively large number of software artifacts are produced as a result of following a rigid, elaborate, detailed planning process.[7] Heavyweight processes also suggest a large development team and a plan that attempts to predict the activities of the team over a relatively long time span. These processes have historically used a "waterfall", or sequential, lifecycle.

The Rational Unified Process (RUP) is a well-known process among the formal, traditional software community.[8] It defines a large set of roles, activities, and artifacts to be produced during the software development process. In practice, no group would implement all of RUP's activities and artifacts but would instead tailor RUP for their particular needs by selecting appropriate roles, activities, and artifacts. RUP has some guiding philosophies best described by its six Best Practices,[9] several of which are emphasized by the course described in this paper:

Develop Iteratively

Rational advocates that software should be developed iteratively instead of using a traditional waterfall lifecycle. Developing iteratively reveals issues earlier, thus reducing a project's risk profile. This consequently increases confidence and expectations in both the developers and customers.

Manage Requirements

Software projects commonly demand requirements to be gathered from several analysts as well as from several sources including customers, end users, and domain experts. Collecting, organizing, and prioritizing these requirements is a large task that can determine the success of a software project.

Use Component-Based Architectures

Software projects should be built using several new or existing well-defined modules, called components. Well-designed component-based architectures are easy to understand, they promote reuse, and can adapt to change.

Model Visually

RUP uses the Unified Modeling Language (UML)[10] extensively to document requirements, analysis, architecture, design, and deployment decisions.

Verify Quality

Testing for software quality is built into the process at each stage rather than being viewed as an afterthought. Quality testing is performed by developers rather than by a separate group after completion of the project.

Control Change

Software projects contain name artifacts, requirements documents, design document, code, test plans, test cases, etc., all of which will evolve over the lifetime of the project. Configuration management systems are used throughout the industry to manage these changes and provide shared access to artifacts.

Lightweight (or agile) processes and XP

A *lightweight* (or *agile*) process is typically thought of as a compromise between two extremes: no process at all, and a cumbersome, bureaucratic, heavy process.[11] The problems with following no process at all should be self-evident. Two of the significant problems seen with following the heavy processes are: they resist change and they are process oriented. The heavyweight processes produce a large number of documents and make long-term predictions and plans. When things change, many documents and many long-term plans must also change. Because this is difficult, the heavyweight processes tend to resist change. In addition, the heavyweight processes focus on designing a general process that will work well regardless of who uses the process. However, the background and skills of each development team vary, making it difficult to define a single, all-purpose process. The lightweight processes have attempted to meet both these shortcomings. In particular, lightweight processes embrace the change that is inevitable in software development and they are people-centered (rather than process-centered).

The eXtreme Programming (XP) process has emerged as the dominant name in the agile process movement.[12] XP (and agile processes in general) are unique in that, unlike other software processes that emphasize non-coding activities, XP's focus *is* the coding activity. All other activities are structured to make that activity produce higher quality and more productive results. Several of the XP best practices emphasized in the "Software Factory" class are described below:

Continuous Integration

XP advocates integrating and building the system several times a day. As a result, integration problems are caught early on, making the problems easier to locate and resolve.

Planning Game

The planning game is XP's project management activity. Developers meet with customers to prioritize requirements, estimate resources, and produce an iteration plan.

Simple Design

The system design should be as simple as possible so that it is easy to understand, implement, and change.

Testing

Programmers as well as customers write test units. Software is tested incrementally and frequently, and tests must be passed before development can continue.

Collective Code Ownership

Anyone on the development team can modify any part of the system at any time. Thus, there are no delays in making changes while waiting for others to submit important modifications.

Refactoring

Refactoring is rewriting code to remove duplication and to simplify the code. As code is added incrementally, redundancies and complexities can result. When changes occur and a broader understanding of the system is reached, code often needs to be rewritten to simplify the implementation and improve the architecture.

XP is particularly well-suited for high-risk projects and relatively small development teams (2-10 people). Since capstone projects typically involve small teams, XP is a logical software process. In addition, XP advocates short iterations – two weeks. This allows students to experience several iterations even in a single semester.

It should be noted that RUP can be thought of as a lightweight process. RUP is a tailor-able software process, and Rational has predefined several road maps for development domains like component development and e-business. In fact, one could tailor RUP to the point that it describes one of the agile processes. However, doing so would remove some foundational RUP philosophies where RUP and XP disagree. For example RUP is not comfortable with the practices of Collective Ownership, Continuous Integration, and most importantly producing software architecture through constant Refactoring.

3 CET 415 Course

The Division of Computing Studies (CST) at Arizona State University created the applied software process course titled "Software Factory" (CET415) in the Fall of 2001. The initial purpose was to provide a more practical perspective on software development than the lifecycle approach taken in many traditional software engineering courses, including the Software Engineering course within CST that is prerequisite to the "Software Factory" course. The "Software Factory" course has students work in teams to solve computing-related problems for real customers using tools and techniques advocated by two software process used in industry, RUP and XP. That goal has also led to applications being used to run business within the organization itself, both inside and outside the department. This section discusses the motivation behind this course, the structure and content of the courses, and results from teaching it for three semesters.

3.1 CET 415 Objectives

This section discusses CET 415's objectives regarding the student's experience with managing requirements, communicating with developers and customers, releasing versions of software, and performing project planning and management.

CET 415 uses projects from real customers. While using real customers is not in itself innovative, it is a vital, mandatory component for an applied software process class. Software development requires communication among many different stakeholders, among them developers, customers, end users, project managers, and removing one of those roles limits the experience. Customers for the class were taken both from inside and outside the university. Some projects have carried over across semesters and one, the Microelectronics Teaching Factory, has continued for three semesters.

CET 415 also uses tools found in industry. Software developed for use in any organization must be treated like an asset to the organization. Software resulting from student projects must therefore be developed to the degree possible as an asset developed professionally. Configuration management and unit testing frameworks are widely used across industry but are not used in most, if any, other academic projects. CET 415 students are required to maintain their system in a CVS repository and to create automated tests using JUnit or similar framework (e.g. NUnit for the .Net languages). Students also use Rational Rose to create diagrams and models of their systems and manage those documents in CVS as well.

In contrast, projects that leave students to their own devices regarding tools provide an extremely poor model for software practices. Quality software processes enforce configuration, build, and release management. In other group projects the authors have seen, students at best use a Yahoo Groups account to share source code and other artifacts or, more commonly, simply email their code to each another. Without providing students with techniques and infrastructural tools, we are breeding poor software practitioners.

Finally, CET 415 also requires artifacts and activities from both formal (RUP) and agile (XP) industry processes to help students be as productive as possible in their 16-week effort. While it is not possible to simulate a real RUP or even XP project in a semester course, each team is required to produce essential artifacts and follow essential practices from the respective processes. Both processes advocate iterative development so each team must release software in three one-month increments. The teams must also perform XP's User Stories, Planning Game, and measure their team's Velocity. Several teams also perform and document Spike Solutions by storing the prototype code in CVS. Regarding RUP, students are required to produce an Architecture Document, a Use Case Diagram for their project, and a Use Case Realization for one scenario of their system. Details and rationale behind these artifacts and activities are discussed below.

3.2 CET 415 Structure

The students in the class are partitioned into teams of three to five students. Teams must be small enough so that each individual has sufficient responsibilities but large enough to experience communication and coordination issues. Teams are required to produce three iterations and deliver iteration status reports in class. Their first presentation occurs two to three weeks into class to present the project's goals and objectives and the team's iteration plan for the semester. This is followed by 3 more presentations roughly three to four weeks apart for the iteration status updates.

One challenge is to get students information early so they can become productive enough to produce something that executes by their first iteration, as advocated by incremental methods like RUP and XP. The class begins the first week with a CVS assignment so all teams have created a repository and all team members have checkout, modified, and committed changes to it. The class then focuses on XP requirements and planning practices – User Stories and Planning Game. The XP community has several workshop exercises that are good in-class activities.[13,14] This understanding is vital before releasing them to talk to customers to gain and manage requirements. Other XP practices are discussed in the class and used throughout the semester – Small Releases, Collective Code Ownership, Unit tests, Continuous Integration, Refactoring, while others – Pair Programming, Stand Up Meeting, On Site Customer – are problematic since students do not work on these projects all the time and have schedule conflicts. Although, one team did write all their code in pairs and were very positive about the practice.

RUP is introduced mid semester, around the time of the second iteration. RUP artifacts students are required to produce and manage in CVS are a Use Case diagram (no Use Case Specifications) and an Architecture Document containing a Deployment Diagram, Component Instance Diagram for the executable components in the system, and Realizations (one Class Diagrams plus interesting Sequence/Collaboration Diagrams) showing the objects they use to solve at least two design mechanisms within the system. As an example, the most common mechanisms are the design solutions for distribution.

4 CET 415 Projects

This section discusses the projects created by students in CET 415 used within ASU. Those discussed in this paper are listed in the table and detailed in the section below.

Project	Date Started	Customer	Status
Microelectronics Teaching Factory	8/02	College of Technology	Deployed and in use
Graduate Admissions System	8/03	Computer Electronics Department	Completed, will begin use in 1/04
True Outcomes/Blackboard Integration	8/03	Computer Electronics Department	In progress
Word Analysis in Print Media	8/03	College of Business	Deployed and in use

- Microelectronics Teaching Factory (MTF)

The MTF is an NSF-sponsored curriculum development effort in conjunction with community college faculty. ASU is working with the local community college system to provide access to a clean room. This grant is funding curriculum development for lab exercises within the clean room to augment existing community college courses. Those curriculum developers must collaborate on sharing documents and put those documents through a workflow that follows a develop, review, put-into-production process. The web-based system must archive each version of the curriculum document and make it available via the web. The MTF group looked at existing document management systems and, due to their cost, elected to try a CET 415 project to see if students could produce a suitable system.

The project has continued for two separate semesters and one summer with an independent study student. The first effort was only mildly successful, but the customer became more involved in the second and the results were much better. The NSF grant purchased a server that must be administered. New requirements are being added for Spring 04 and will be supported by an independent study student.

- Graduate Admissions System

Our department's current graduate admissions process is executed with paper forms routed between committee members, chairs, and administrative assistants. The recommendation process requires access to various web sites to view applicant information, which are scanned electronically. To improve both the productivity and visibility of bottlenecks in the process, this project created a uPortal[15] *channel* to be installed into the university-wide uPortal instance run by the IT organization. Channels (i.e., portlet) are small portal applications that run inside the uPortal container. The application manages student information (name, ID, program applying to, etc.), committee recommendations and comments and routes the virtual document to the next person in the workflow chain. It also provides links with automatic logins to the appropriate applications for viewing applicant information.

The project was successfully completed in Fall 03 and will be used in Spring 04. To support an enhancement and perform bug fixes during the Spring rollout, a student (the same performing MTF enhancements above) will provide support. The IT organization has also become less interested in hosting student projects on the university portal instance, so we may need to modify the entire application to a servlet-based web application to deploy on the department's web server.

- True Outcomes/Blackboard Integration

Our College has adopted a product True Outcomes (TO) for helping with outcomes-based assessment, while the IT department uses Blackboard (Bb) for course management. Instructors, like the instructor for CET 415, use Bb for assignment posting, submissions, and grades. However, this information must be reentered into TO to manage the assessment process. This project takes a Bb course export, parses it, and then imports assignments, grading criteria, and scored submission into TO for later retrieval with outcomes-based assessment.

The project required True Outcomes to add an XML-based import utility and for their team and ours to agree on a common XML format. This project was initiated in Fall 03 and is not yet complete, which is not surprising due to its scope. The exporting and packaging of the Bb data is completed (hopefully) and the import into TO needs to be implemented. The solution also has expectations on the Bb instructor to name assignments consistently and place assignments and grading criteria in a specific Bb location so the data can be automatically parsed.

- Linguistic Analysis Tool

This application supports a faculty member performing research in linguistic analysis by observing words patters in media articles. The algorithm gathers words in thousands of articles and watches for the trends of related words and measures influential words in areas of interest. This application was the first and only CET 415 project used outside the College and therefore the authors have little knowledge of its use. More information can be obtained at the projects web site.[16] The project completed successfully and is deployed and in use.

All the projects above from a professional perspective would have been mildly successful. They were completed, but with less functionality than anticipated. The resulting architectures range from acceptable to poor complicating future enhancements. The enhancement problems can be seen in the MTF project, as it was the only one to span multiple semesters. Students in the second semester wanted to throw out the initial effort and start from scratch. There is a large "not invented here" attitude among students where they are less interested in modifying and enhancing someone else's code.

5 Project Success and Issues

There are several ways to view success for these projects. From an academic perspective the projects have been extremely successful as teaching tools for students. The course is highly rated by students who like the use of practical techniques and tools and creating programs used by real people. To measure the success of the projects themselves one must consider the customer's satisfaction and the resulting technical quality of the software.

Customer satisfaction has been good at best. The projects have met the minimum customer expectations. However, the students are not trained at managing customer relationships nor are they experts at gathering requirements and understanding customer needs. So, much of this problem is beyond the scope of the class. Another issue for customers is there is not a continual development team working on their project as there is 100% turnover ever 3 months on the teams and then downtime between semesters and during the summer.

The technical quality of the software varies, but in general it is terrible. Students do not make the connection between software as an asset from software thrown away after the TA executes it once. Production software that will have a long life must be developed in a quality manner. Viewing the code, it is obvious students take needless shortcuts to simply save themselves time. The authors feel poor quality code is an epidemic problem with the educational computing community and hurts the entire profession. Below is a list of specific problems discovered in three semesters teaching CET 415:

- Limited customer relations skills

Engineering and Technology students are not trained nor conditioned to interact with customers. In practice, a customer's interface is a sales person or account manager, not a programmer. So, it is not surprising customers will be less satisfied as the problems and issues are not being "spun" appropriately for their digestion.

- Team turnover and work environment

Project success requires them to not loose key people and provide the team with a productive work environment. Class projects turn their entire staff over every semester. This problem is almost insurmountable to ensure any type of quality product. The problems and issues of getting an entire new set of developers up to speed on a system, teach them the necessary skills and tools, and then have them make any significant enhancements in a 4 month time frame is the largest problem faced with the effort.

With respect to the work environment, there is none. Students work at independent times and locations complicating communication. Work environment is an important factor in productivity and the environment given students is near the bottom of the productivity scale. One solution would be to reserve some lab for 4 hours and require students to attend the "lab" and use that time to develop software in a more realistic environment.

- Project management

As with most projects, success is commonly dictated by how well the project is managed. Ideally the course instructor would act as the project manager and be intimately involved with each project. In practice, though, the instructor will not have time and therefore serve more as an advisor or consultant. Students must therefore manage their own projects. Using XP's User Stories and Planning Game have been effective tools, but weak project management is a reason for limited project success.

Another problem with project management is having teams understand what features to develop in what order. Project management chooses feature based on customer value, cost to implement, risk to implement, etc. Several projects did not complete important features because they waited too long to being them, mainly because of risk. In both cases the feature required the students to engage another organization to either deploy the resulting application or to interface to their product. Those conversations must start in the beginning of the semester not the end.

Another problem with students managing their own projects is not understanding when to elevate an issue. Students need to understand the instructor also plays the role of their manager. There are issues that are beyond their control that require the instructor to get involved. However, some student view that as a weakness on their part that they cannot get the project completed without help. When in reality, they need to raise an issue to their "manager".

- Student skill sets

Skills of students and consequently teams vary widely. Our department is new and has only been teaching a software-based curriculum for three years. So we expect the student skills to rise significantly as the department becomes more mature. However, a few bad students can add significant errors to a system. XP's pair programming or, at a minimum, requiring all software to be peer reviewed can help solve this problem.

6 Conclusions

Students can provide an inexpensive labor source to create application to help run academic business units. We have discussed some projects developed in the "Software Factory" course in Computing Studies at Arizona State University East. The course provides an outstanding academic opportunity for the students. Students learn more when developing real applications used by real people and enjoy the experience. The quality of the resulting products can vary dramatically, however. We cited several issues that can lead to reduced quality including the student's technical skills, their skills at project management, their skills at software development methodology and process, and finally the turnover of students every semester for long running projects.

The value to the academic unit can be directly attributed to the project management effort for the project, either internally by the team or by the instructor. On poor teams, many of the above issues can be mitigated by direct involvement of a qualified instructor. Instructors with software project experience can mentor students on not only technical aspects, but more importantly procedural issues like interacting with customers and end users, performing builds and deploying versions of software, scheduling and planning, etc. Therefore, more critical applications will need more time and involvement from an experienced instructor.

Bibliography

1. Granville Miller, "Sizing Up Today's Lightweight Software Processes" June 2001. http://www.computer.org/itpro/homepage/May_Jun01/miller/
2. http://www.abet.org/criteria.html
3. R. Beasley, "Conducting a Successful Senior Capstone Course in Computing," Consortium for Computing Sciences in Colleges, 2003.
4. http://www.cse.msu.edu/~cse498/
5. http://www.cs.southwestern.edu/~owensb/SE/
6. C. Larman, "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd ed.," Prentice Hall, Inc., 2002.
7. P. Kruchten, "The Rational Unified Process – An Introduction." 2nd edition, Addison-Wesley, 2000.
8. "Rational Unified Process: Best Practices for Software Development Teams," Rational Software White Paper TP026B, 2001.
9. M. Paulk, et. al., "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1993.
10. G. Booch, I. Jacobson, and J. Rumbaugh, "The Unified Modeling Language Users Guide", Addison Wesley Publishing Co., 1998.
11. Fowler, M. "Put Your Process on a Diet," Software Development, CMP Media, December, 2000.
12. Beck, K. "Extreme Programming Explained – Embrace Change, " Addison-Wesley, 2000.
13. http://www.xp.be/xpgame/
14. http://csis.pace.edu/~bergin/xp/planninggame.html
15. http://mis105.mis.udel.edu/ja-sig/uportal/
16. http://www.crawdadtech.com

HARRY KOEHNEMANN

Harry Koehnemann is an Associate Professor in the Division of Computing Studies at Arizona State University East. He completed his Ph.D. from Arizona State in 1994. His research interests include distributed web-based software systems, software process, and network-enabled embedded devices. Please see his home page for more information and his vita at http://latitude.east.asu.edu.

BARBARA D. GANNOD

Dr. Gannod is an Assistant Professor in the Division of Computing Studies at Arizona State University East. She completed B.S. degrees in Mathematics and Computer Science and a Secondary Education degree at Calvin College in 1992. She received her Ph.D. in Computer Science from Michigan State University in 1997. Her research interests include engineering education and high-performance computing.