Preparing Computer Science Students to Question and Improve a Software Development Process

Carol Wellington

Abstract - Shippensburg University offers a B.S. in Computer Science degree with a concentration in Software Engineering. In addition to a core of computer science classes, this concentration contains three four credit software engineering courses. The main goal of this concentration is to prepare the students to be able to measure the software development process and enact appropriate modifications. This requires that they must be aware that there are a variety of software process alternatives and when various aspects of each process might be beneficial. In addition, they need to be able to apply appropriate statistical analysis to assess the process. In order to accomplish this, our Software Engineering concentration includes three software engineering specific courses: Traditional Life cycle, Testing & Extreme Programming, and Software Metrics/Project Management. In the first two courses, the students work in teams to develop products for customers using two disciplined, but very different, processes (a variation of TSP[3] and XP[6]). By comparing these experiences, the students learn about making the choices that software engineers have to make regularly. In Software Metrics, we study methods of assessing risks, the process, and modifications to the process. This paper will describe these courses and the results of our assessment of these courses.

Index Terms - software engineering, curricula, Extreme Programming

INTRODUCTION

In addition to having the skills of a computer scientist, software engineers need to be able to define an appropriate software development process and refine that process with the goal of continuous improvement in quality and on-time delivery. With the advent of agile software development processes (Extreme Programming (XP)) as opposed to traditional development processes (waterfall, V-model, etc.), software engineers have many options to consider and need to be able to compare and contrast them to decide which aspects of these processes are appropriate for a particular project.

Most software engineering programs include a capstone course where teams work on large projects using a traditional development process. For Shippensburg University of Pennsylvania, this has historically been a two semester sequence. In order for our students to have experience with multiple processes, we have modified this sequence. In the first semester, the teams use a traditional development process and in the second course, they use Extreme Programming (XP). The goal is to have them immersed in two very different (though equally disciplined) processes to allow them to discover the effects of alternative process strategies.

SOFTWARE ENGINEERING CONTENT IN CORE COURSES

The core of our computer science program focuses on the topics advocated by the ACM [4] so that all of our students have a strong computer science background. Similarly, we believe that all of our students need exposure to software engineering content and many of those topics are also included in the core courses. We adopted an “Objects First” [3] approach to teaching our students to program. As in the ACM guidelines, our programming sequence consists of three four-credit courses and includes topics in UML, software development life cycles, responsibilities of practitioners in developing code, integrated development environments, CASE tools, testing, and self-documenting code. Similar topics are often included within the context of courses like Operating Systems and Computer Organization.

SOFTWARE ENGINEERING SPECIFIC COURSES

While we have followed the development of the Software Engineering Body of Knowledge [2], the size of our concentration does not allow us to address the bulk of that material properly. We chose instead to focus on the Core and Foundation areas recommended by SEI [1]. The main goal of our software engineering concentration is to expose our students to multiple development processes and project management strategies. This allows our students to begin to compare and contrast such paradigms and understand how to select a process for a given project/organization. We also begin to address the concept of continuous improvement and process assessment. As the drafts of [5] have become available, our concentration, though limited in scale, supports a number of the student outcomes required of software engineering curricula.

Due to the size of our core, the software engineering concentration is limited to these courses: Networks, Traditional Life Cycle, Testing and Extreme Programming, Software Metrics, and two computer science electives.

---

1 Carol Wellington, Associate Professor and Chair, Computer Science Department, Shippensburg University, Shippensburg, PA 17257 cawell@ship.edu

0-7803-8552-7/04/$20.00 © 2004 IEEE

34th ASEE/IEEE Frontiers in Education Conference

October 20 – 23, 2004, Savannah, GA
In this course, the students work in teams to develop an initial release of a product for an outside customer. They use the traditional process of requirements analysis, high level design, detailed design, implementation, unit test, integration test and system test. While we generally follow the Team Software Process [4], we discuss similar alternatives like the V-model and Boehm’s spiral model.

Throughout this process, they use appropriate tools including CASE tools, UML, configuration management tools, and integrated development environments. By the end of the semester, a working prototype is available for customer review. Throughout this process, the students make estimates, plan, measure their progress and assess their process with metrics. A subset of the students in each section of this course will have completed the software metrics course and will be applying those techniques in their teams in this course.

The projects in this course vary depending on the interests of the students. However, there is always a requirement that the project must have an outside customer. This gives the students experience with interacting with a customer and helps ensure that the projects are real world projects. We find that this gives the students more motivation and makes them behave more like they are employed as opposed to students in a class. Recently, the projects have included a system for campus security officers including a dispatcher station and Palms for the officers, educational games, a rental property management system, a web/palm application for faculty and students to manage class assignments, and a system to help evaluate stock buying and selling strategies.

In this course, we have varied the size of the teams and the complexity of the projects. Originally, our teams consisted of four to five students. Unfortunately, we found that projects that were small enough for a team of this size to handle did not really demand the rigor of software engineering techniques. For example, teams may not have needed to use the configuration management tools since most of the coding was being done by two or three individuals. For those teams, the tools seem to be cumbersome with little benefit.

We then raised the number of students on a team to about eight. This was big enough that the students began to see the need for CASE tools and configuration management tools. Unfortunately, teams of this size required more significant project management skills. Since our students are generally inexperienced in this area, a number of teams poorly managed themselves and, therefore, were unsuccessful. This hindered their understanding of how a project using this development cycle should function.

This year, the entire class is working to develop a single product with teams of six to eight in five subsystems. Clearly, software engineering tools and techniques are required for this project. Since the teams now have to work together, they manage themselves, but project management is provided by the instructor. This allows the instructor to mentor the team leaders and support them as they develop project management skills. This situation has truly motivated the necessity of software engineering and project management techniques.

Testing and extreme programming

This course is very much like our Traditional Life Cycle course in that the students are working on teams to develop a project. In fact, they are usually making subsequent release for a project developed in the Traditional Life Cycle course. However, now the development process in Extreme Programming.

In the beginning of the class, strategies for development of automated unit/integration tests are studied. In particular, the students learn to develop tests using an automated environment like JUnit within an IDE like Eclipse. Strategies for developing useful tests like border case and worst case techniques are practiced.

Using this testing framework, students proceed to extend their projects using Extreme Programming techniques. From code development perspective, there are two significant changes from the Traditional Life Cycle course. First, the students must program in pairs. We practice selecting partners with criteria like experience with a particular technique or responsibility for code that interfaces with the code being developed. We also talk about sharing the keyboard and research activities while working together on algorithms and strategy. The second significant development change is that the pairs must develop the automated test cases before they write any code. The goal is for them to develop an iterative strategy where they develop a few test cases, write the code to make those pass and then write the next set of test cases. This is contrary to their training and requires significant discipline at the beginning, but pair programming helps them stay focused.

The project management strategy in this course also matches that used in Extreme Programming. The course is 14 weeks long, so we have 2 weeks of introduction and then four iterations lasting three weeks each. In each iteration, we use the Planning Game with our customer to set the contents of the release. They work with their customers to develop, prioritize and schedule Story Cards so that they see how that strategy structures the relationship between project management and the customer.

The teams then use the Iteration Planning Game as their project management strategy for that iteration. For each Story Card, they develop a set of Task Cards to plan the activities required for the iteration. Within the Iteration Planning Game, they plan who will be responsible for each task and how the work will be completed during that iteration.

As in the Traditional Life Cycle, the students track and assess their own activities with support from the instructor. This includes setting their project velocity and tracking metrics that they deem to be important at each iteration.
Software Metrics/Process Management

While the enhancement of our capstone course to include Extreme Programming has been valuable in making students aware of different processes, software engineers need to be able to assess a process and measure the effects of changes to the process as part of continuous improvement efforts. In order to address this, we are adding a course in software metrics and process management. This course explores and evaluates current methods of measuring the software development process. The need for such measurement is motivated by application of the Personal Software Process (PSP) [3]. PSP is a structure for measuring personal software development activities and motivates how metrics can be used to predict future activities. Techniques for gathering and analyzing common metrics are used to support process assessment and improvement.

Assessment

In order to assess our students’ analysis of these software development processes, they write a comparison paper at the end of Testing and Extreme Programming. At this point, they have experienced both development courses and we can assess how well we have met our goals.

In assessing these papers, we are assessing not only our students’ understanding of the processes they experienced, but also their ability to analyze the value of the various processes. For this, we are looking for statements about the positive and negative effects each process had on their experience and on the quality of the resulting product.

Finally, we have designed this sequence with the goal of helping the students see the importance of evaluating the development process. While we do not specifically address it, the students should begin to see that aspects of various processes can be combined and modified to meet the needs of a specific situation. Therefore, our assessment is also looking for comments relating to process selection and continuous improvement. The following sections demonstrate the results of our assessment including analysis of the proportion of students who provided specific types of comments and quotations from these student papers.

Alternative Processes Should Be Considered

Our primary goal is for our students to see that alternative processes can be appropriate. The first part of this that we look for is exactly what one student stated:

- “Extreme Programming is a deliberate and disciplined approach to software development”

It is critical that they understand that, even though XP does not produce the same documentation and is not managed in the same way as the traditional processes, it is still a disciplined, measurable process.

After we have established that there are alternative processes, the next step is for the students to recognize that no process is appropriate for all situations.

- “A project manager shouldn’t be limited to these two options, however, as different groups of people react differently to each model.”
- “There is no Holy Grail method that every organization should use.”

Our analysis of the papers showed that 87% of the students made explicit comments reflecting the fact that each of these processes was appropriate in some situations.

Comparison of the Structure of the Process

The most basic comparisons that the students can make relate to the obvious differences in the structure of these processes. We look for statements about the effects of these changes in structure. Fully 98% of the students made comparisons between the structures of the process (one student did not seem to understand the point of the paper).

When discussing the strategy of writing unit tests before writing code in Extreme Programming, students see a number of effects. Common effects include:

- “Writing tests helped me think about the project and slowly allowed me to think more in depth about how to do it.”
- “Writing code for the tests allowed us to understand what we were writing for and made coding much easier.”
- Developing automated tests is “a kind of planning in itself. It makes the programmers look ahead to the task.”
- Developing the test cases first “helped dramatically to determine areas where our program lacked robustness.”

These comments demonstrate that the students are attributing appropriate effects to the strategy of writing tests first. It is important to note that not all of the comments are positive. One student stated,

- “As the number of iterations increases, so does the complexity of the automated testing.”

This student believed that the effort required to develop appropriate new tests would become overwhelming as the project grew large. Given the tools and strategy we used, he is probably correct. Unfortunately, he did not propose tools or strategies that could relieve this complexity.

Students also comment on the quality that resulted from the testing in extreme programming:

- “Code quality improved during the extreme programming portion.”
“By testing as you go, you find small bugs faster and earlier which can eliminate bigger bugs in the future.”

The students also saw dramatic differences from the pair programming in extreme programming:

- “When we were using the waterfall model, we worked alone and had little idea what was going on in other portions of the code.”
- “Pair programming reduced the need for outside meetings.”
- “Since you are programming with a partner, you spend less time debugging and working out algorithms than you would if you were alone. You also have less stress from not being able to figure out why particular code is not working.”
- “Defects in code were much lower using the process of Extreme Programming since each member had someone paired with them during the coding process”
- “Extreme programming mitigated the entire stress fact because it emphasized constant teamwork.”

The students clearly see the role of communication within a team and that two processes used different strategies for how to achieve the necessary communication. In addition, they see how good communication can impact quality of the product and the stress level of the engineers.

While the students seemed to like pair programming, they do see situations where it seemed wasteful:

- “Pair programming was unnecessary when we had easy parts of the project to do. . . The other person could have been pounding out another set of easy code and we could have gotten more done.”

This student is thinking about the efficiency of the development process and understands the cost of pair programming. This shows that he has learned to question the process even when that aspect of the process is popular. However, he did not take the second step and offer strategies for how to determine when pair programming is inappropriate.

Finally, since the students have had to make several releases of their project, they understand the challenges of maintaining good code. That allows them to appreciate the value of refactoring:

- “By having better-written code, it is easier to make changes in the future because you can follow what is going on.”

Non-Technical Effects of the Process

Comparing the mechanics of the process is only the first step a software engineer needs to make. After that analysis, we are looking to see if the students compare non-technical effects of the process. This includes things like morale, productivity and stress level. The following examples show that the students do begin to make these comparisons:

- “There is a measure of perceived productivity with XP that I do not think you get with the traditional life cycle.”
- “With traditional software development, I got a feeling that I was hacking away at a huge block of wood, and my programming attempts were only splinters of the block.”
- “XPs stress level is more consistent than the traditional life cycle.”

In fact, 77% of the students commented on the difference between the stress levels they experienced during the two semesters. While most felt less stress during XP, there were three students who felt less stress during the traditional process. A lot of the students attributed almost all of the planning process and its relationship to the development process. The most prevalent aspect of this that they comment on is the difference between the longer term planning of the traditional model and the shorter term iteration planning of extreme programming:

- “Planning a long term schedule is extremely difficult. There are always unforeseen issues ... that have a larger impact.”

In addition, they begin to appreciate the value of measurable milestones that are relatively close together:

- “Deadlines that are far apart tend to cause a ‘last minute scramble’ to meet the deadline.”

Comments about the different effects of long term versus short term planning we made by 57% of the students

Finally, the planning of individual assignments in the tradition life cycle class is predominantly left to the students. We were concern that, with their lack of experience in negotiating such things, some students get assigned to tasks that do not match their skills and interests. However, only 23% of the students stated that the ability to pick a task was important in XP. We attribute this to the fact that most students did not feel that assignments or deadlines were set for them under either process. However, when we use the iteration planning game in XP, some students do appreciate the ability to pick tasks and set their own target for the amount of effort a task will take:

- In XP “since you get to choose what tasks/stories you do, you should be more interested or capable of creating code for the tasks that you have chosen.”
difference to the process (which shows they see the effects a process can have). However, some of that distinction could also be a result of the fact that they used XP second. At that point, they had been working with their teammates and on that project for more than a semester, so some of the difference in stress could be attributed to experience.

Process Improvement

While not all students get there, our goal is to help the students think about process improvement in the context of a particular organization. Since they had not yet had the metrics course, we were pleased that 36% of the students offered suggestions for how the processes could be combined or altered for particular situations. This focus on process improvement is exactly what a good software engineer needs. While there are too many challenges for our students to see them all, some of our students begin to see how process improvement fits within an existing organization:

- “For a company that is currently using traditional methods for project management, the best thing they could do is try some of the ideas behind extreme programming. They would need to try only one idea at a time. After a set period, they should then reevaluate their methods to see if what they are trying improved the product, morale, and productivity

It is important to note that these comments also show that the students have a definite preference for Extreme Programming. Since these courses have been taken in sequence, part of their comfort may come from the fact that they have been working with the same team on the same project and are more experienced in the second semester. In the future, these courses will no longer be a sequence. At that point, it will be interesting to assess which process they prefer and how their experience differs if they schedule the Testing and Extreme Programming course prior to taking the Traditional Life Cycle course.

Also, as of the writing of this paper, the software metrics course has only been offered to graduate students so we will assess its impact on our undergraduate students over the next year.

CONCLUSIONS

This paper described the software engineering concentration in our Computer Science bachelors’ degree. This concentration includes two courses in which our students apply two very different processes to develop products for outside customers and a course in software metrics and project management. The primary goal driving the structure of these experiences was to increase the students’ awareness of the range of alternative processes and the varying effects those processes have on the development environment, on-time delivery, and product quality.

To assess these courses, we analyzed papers our students wrote comparing their experiences in the two product development courses. Our assessment shows that almost all of the students develop the ability to compare and contrast the technical and non-technical effects of the process. As a result, all of our students understand that alternative processes should be considered in the context of a particular project and organization. In addition, some of the students begin to see the value of assessment and continuous improvement activities. As we add the third course in this sequence (Software Metrics/Process Management) we hope to raise the number of students meeting the goal of valuing process improvement and assessment activities.

We have demonstrated that this software engineering concentration, in addition to a strong computer science core curriculum, provides our students exceptional experience, and should allow them to be leading software engineers in the companies that hire them.

REFERENCES


