A Software Engineering Course with
an Emphasis on Software Processes and Security

Richard G. Epstein
Department of Computer Science
West Chester University of Pennsylvania
West Chester, PA 19383
repstein@wcupa.edu

Abstract

This paper describes an undergraduate course in software engineering which introduces
students to a range of approaches to developing software. This course is a required
course in the Computer Science major. Special emphasis is given to PSP, CMMI, agile
processes, and Open Source software development. Students work on a team project
which involves developing a software process for a pretend company. The course also
devotes attention to the strengths and weaknesses of various processes in terms of
developing secure software. Students are asked to address security concerns in the
second of their two team projects. Serving as a basic introduction to software
engineering, this course also covers important issues relating to professional
responsibilities, ethics and work culture. In many ways, the course introduces students to
the human dimensions of engineering software.

1. Basic goals of this course

The author has designed a software engineering course for an undergraduate program in
Computer Science that emphasizes diverse software processes. Students work in teams to design
a software process for a pretend company. The course emphasizes the fact that there are diverse
software processes and software development environments “out there” and that an organization
often needs to develop a process that is appropriate for its development environment. The basic
goals in this course include: (1) Introducing students to a variety of software processes that are of
use in industry, (2) Giving students the opportunity to work in teams to design their own software
process for a pretend company, (3) Introducing students to ethical and work culture issues in
software engineering, and (4) Introducing students to security issues in software engineering.

The software processes covered in this course include: (1) the Personal Software Process, (2)
agile processes (with an emphasis on eXtreme Programming and Scrum), and (3) Open Source
software development. In addition, students are given an introduction to the Software
Engineering Institute’s Capability Maturity Model Integrated, which while not a software
process, is an important model for evaluating the effectiveness (or, “maturity”) of an
organization’s software process. The students are given a general introduction to security issues
in software engineering and considerable attention is given to how eXtreme Programming might
be improved in terms of security assurance. Several authors have criticized XP and agile
processes more generally in terms of their ability to produce secure software. So, an important
issue in the course is whether agile processes can be modified to include good security practices.
The following sections give more details regarding the topics covered in the course, the nature of the two team projects that are assigned to the students, and the resources that are used to introduce security concerns into the course.

2. Software processes

The focus at the beginning of the course is on software processes. We begin with a general introduction to software engineering followed by a discussion of Fred Brooks’ classic paper “No Silver Bullet” [1]. This leads into an intense two week introduction to the Personal Software Process (PSP) developed by Watts Humphrey [2] at the Software Engineering Institute (SEI). The purpose of this intense introduction to PSP is to introduce students to an example of a documented software process. Although there is no team component in PSP (as opposed to the SEI’s Team Software Process, TSP), PSP does give the students a basic introduction to important ideas for software engineering. These include project planning, collecting data on time spent on various activities, and defect tracking. Humphrey’s ideas regarding quality assurance are quite compelling, and industry experiments with PSP have shown that defect tracking is the aspect of PSP that developers are most likely to adopt if given a choice (see Morisio [3]). Although the PSP has had limited success in industry, it is a good way to introduce students to what more sophisticated processes (like TSP) entail. In fact, PSP training is a prerequisite for TSP training in industry.

The course then moves on to agile processes, with an emphasis on eXtreme Programming (XP) (see Beck [4]). This represents an intentional and radical shift from the heavy-duty data tracking of PSP to a highly iterative process that emphasizes collaboration and creativity. We go over XP in some detail, reviewing the major development practices in XP like requirements gathering using user stories, release planning, iterative development, emphasis on simplicity, pair programming, daily stand-up meetings, coding tests before the actual code is written, continuous integration, and the forty hour work week. Our discussion of pair programming in XP [5] exposes students to a recurring theme in the course: four eyeballs are better than two in terms of preventing defects from being in the delivered code. This eventually evolves into the “many eyeballs principle” for quality assurance.

We also discuss agile processes from an industry perspective, studying some papers that discuss the use of XP and Scrum in industry. These papers include the excellent papers by Grenning [6] and Schatz and Abdelshafi [7]. Grenning discusses how he led a project that introduced XP into an organization that had a historical commitment to “big up front design”. He discusses the compromises that had to be made in order to adopt XP to this kind of environment. Schatz and Abdelshafi discuss how the Scrum agile process was introduced into their corporate environment, with the help of one of the creators of the Scrum process, Ken Schwaber.

Back in 1987 Fred Brooks [1] ended his “No Silver Bullet” article with four approaches to software development that he felt were the most promising in terms of addressing the essential problems in software engineering. He referred to these as “promising attacks on the conceptual essence” of software engineering. It is interesting that two of these four promising attacks are important features of agile processes twenty years after he wrote his paper. These promising attacks are (1) requirements refinement and (2) incremental development.

The next subject covered in the course is the Capability Maturity Model Integrated (CMMI) from SEI. We introduce CMMI (or, more precisely, its predecessor, CMM) using an excellent paper by Mark Paulk entitled “Extreme Programming from a CMM Perspective” [8]. This paper accomplishes several things in terms of the goals of this course. First of all, it introduces CMM and basic CMM terminology (e.g., levels of maturity and key process areas). Second, it shows that CMM is not a process but a means for evaluating a process in terms of its “level of maturity”. Third, it shows that XP is a process and thus can be evaluated using the CMM framework. Mark
Paulk, who was involved in the development of the CMM in the early 1990s, shows that XP is not inconsistent with CMM. He explores the strengths and weaknesses of XP in terms of the CMM key process areas. According to Paulk, an organization that uses XP with proper management commitment and oversight could certainly achieve level 2 in the CMM framework and would satisfy some of the key process areas for level 3 (and even one at level 5, the highest maturity level). Ken Schwaber and Mark Paulk have worked together on various projects and presentations. There has even been talk in the agile process community of Paulk’s intention to look into the possibility of developing a maturity model specifically for agile processes (an exciting idea, indeed!).

This leads into a more in-depth study of CMMI, especially from an industry perspective. We use a paper by Bill Pitterman [9] to see how Pitterman’s company, Telcordia, achieved CMM level 5. Pitterman makes some excellent points about the need for a sensible software process, as opposed to mindless bureaucracy, and the need to give developers a sense of ownership for the process, so that the process does not come down as something dictated from above. Our study of CMMI ends with a discussion of the current reality, which sees CMMI being adopted on a large scale all over the world. We see that more and more small companies are buying into CMM, as reflected in the paper by Guerrero et al. [10]. Finally, we study how CMM ideas could be incorporated into a Web development culture without explicitly striving for a CMM assessment (as described in Wiegers [11]).

The final approach to software development that we cover is Open Source development. In recent semesters, due to time constraints (and the fact that we are devoting more and more time to agile processes), we are devoting a bit less attention to Open Source software development. We look at arguments pro and con relating to Open Source and we have a guest lecturer from industry who sees tremendous promise in Open Source development in coming years.

It is interesting to note that almost none of the resources that we use to introduce students to software processes pays much attention to security concerns right up front. The author tries to alert students to these issues early on, in an iterative fashion. However, a more detailed and systematic approach to the security issues are covered later in the course.

3. Professional and ethical issues

Once the students have been introduced to a variety of important software processes, they can begin to work with their teammates with the goal of developing a documented software process for a pretend company. In class, we turn our attention to a discussion of professional and ethical issues in software engineering. The goal is to give the students a more profound appreciation for their responsibilities as professionals working in the area of software development. We introduce the professional and ethical issues by discussing several real-world scenarios that involved major failures in software projects. The first scenario we cover involves the Therac-25 accidents. The Therac-25 accidents involved very subtle programming errors that caused a radiation therapy machine (deployed in hospitals in the United States and Canada) to kill three patients and injure three others. These accidents are discussed in a powerful paper by Leveson and Turner [12]. We then discuss the Confirm fiasco, in which an integrated airline, hotel reservation, and car rental reservation system was never completed because of the incompetence and one might argue the unethical behavior of project leaders (as described in Oz [13]). This leads us to a general discussion of “why software fails” and a major resource for this discussion is an article with that title by Robert Charette [14]. Our discussions concerning the Therac-25 and Confirm fiascos and the broader topic of why software fails naturally lead to a discussion of the Software Engineering Code of Ethics (Gotterbarn et al. [15]). The author has found that it is useful to emphasize that the code of ethics is essentially about helping to establish software engineering as a true profession, a profession worthy of respect.
4. Work culture and quality assurance issues

The course then moves on to discuss a variety of issues in software development, including issues in work culture and quality assurance. The discussion of work culture focuses on the idea of “congruence” presented by McLendon and Weinberg [16]. They define congruence as the alignment between the internal and the external, between what is thought and felt and what gets expressed in behavior and speech. McLendon and Weinberg view blaming as a primary symptom of a software development organization with a poor work culture. In a blaming culture, the focus is on blaming others rather than on the technical problems that need to be addressed. We then go on to discuss a variety of workplace demons (see Epstein [17]) which can have a negative impact upon the work culture. Students participate in a class exercise which involves exploring how specific workplace demons (e.g., arrogance, inflexibility, sexism, lack of respect for the other, laziness, and boredom) can impact a team that is facing a technical problem in software development. We also discuss Steve McConnell’s notion of a “problem programmer” [18]. We discuss how a team can deal with a problem programmer as well as the reasons why a problem programmer might have evolved into such a state of mind.

Quality assurance is the next focus in the course. We look at industry-based studies of quality assurance practices, including software reviews and software testing. In terms of software reviews, we discuss the survey conducted by Ciolek and his colleagues regarding software review practices in industry [19]. Another article that we discuss looks at the conflicts between software developers and software testers (Cohen et al. [20]). This combines the work culture issues discussed earlier and the quality assurance measures. This leads to a discussion of an interesting paper by Talby el al. [21] that covers in some depth the quality assurance measures that were used in a large project that used an agile process. This paper addresses the important issue of how XP can be scaled up for larger projects. In our introduction to agile processes we discussed this as an important issue and we gave some attention to MetaScrum, a scaled-up version of Scrum). This discussion of quality assurance offers a good segue into the security portion of the course, since quality assurance and security issues are clearly related.

5. Issues in security

The course tries to alert students to the security issues from day one. Occasionally, as we discuss software processes and work culture issues and professional responsibilities, we discuss the security concerns that are now central in software engineering. One component of the course is devoted to the security issues specifically. This portion of the course uses a high-level approach to this problem, rather than focusing on specific coding issues for specific languages. This high-level approach involves emphasizing how security concerns should be incorporated into the software development process during each phase of project development, including the requirements gathering, design, implementation, and testing phases of the project. We look at the basic principles of security as applied to software development, including defense in depth, using community resources, and the principle of least privilege. We examine how these security principles might be in conflict with basic software project goals (like user friendliness and timeliness to market). We also look at the role of tools that are available for scanning software for vulnerabilities. The introductory security lectures were greatly influenced by the excellent book by Viega and McGraw [22]. Viega and McGraw give detailed descriptions of how security concerns (like risk assessment) can be integrated into the basic phases of the software life cycle.
Our security lectures then focuses on the relationship between software processes and security. We begin this discussion with the article by Noopur Davis et al. [23] that supports certain processes as being consisting with developing secure software. The paper by Davis et al. is a summary of a report developed by the Security Across the Software Development Lifecycle Task Force organized by at the National Cybersecurity Summit in December 2003. It is interesting to note that eXtreme Programming and agile processes, more generally, are not mentioned in this article at all.

We then turn our discussion to eXtreme Programming and the concerns that some authors have expressed regarding its appropriateness for developing secure software. The goal is to have an animated discussion about how the basic practices of XP could be modified to explicitly incorporate security into the XP framework. In particular, we look at how various XP practices might be strengthened to address specific security goals listed in the paper by Davis et al.

Specific suggestions that often arise in these discussions include the inclusion of a security engineer on the XP development team and also the use of security stories to augment the usual user stories. Students also suggest additional practices for making the XP software testing process more security-oriented and for including security issues in the project planning process. This is just a short summary of the many interesting ideas that arise during this important discussion. Clearly, making agile processes more appropriate for developing secure software is an important topic for software engineers in the era of the Net, and the author hopes that this discussion will turn on the students in terms of thinking about the contributions that they might make to this discussion as they go out into the work place.

6. Team projects

The course involves two team projects. The first of these projects is the one mentioned at the beginning of this paper. Students are asked to integrate ideas from the various processes that we discussed in order to create a defined software process for their pretend company. Most of these pretend companies definitely have a bias in favor of agile processes. In other words, the students seem to be attracted to the agile process approach to software development.

Several months ago, the author received an interesting e-mail from a former student, a recent graduate. He said that his company, which has been noted in the media as a fast-growing company here in the Philadelphia area, has adopted the software process that he and his teammates developed for their pretend company in this software engineering course. Their process was a scaled-up version of eXtreme Programming that had some similarities to MetaScrum.

The final team project, delivered at the end of the semester, involves an in-class team presentation which is intended to synthesize ideas from the entire course, including the concerns relating to building secure software. The emphasis for these in-class team presentations is on creativity. While some teams end up giving a rather standard PowerPoint type of presentation (and some of these presentations have been excellent), others take advantage of the opportunity to present a dramatic presentation in class. These dramatic presentations are either performed in class or are presented as a video featuring the members of the team. Some of these dramatic presentations have been truly remarkable and inspiring. For example, several of the dramatic presentations have told the story of a company that has migrated through a sequence of software processes. One team went out into industry and interviewed a software engineer about his specific practices. The interview involved twenty-nine questions that the students prepared and the author was struck by the quality of those questions.

7. Conclusions

The author has gotten excellent feedback from students, especially to the effect that this coverage of software processes turned out to be very helpful when they went out there for job
interviews. An important aspect of the course is to acknowledge that security is becoming more and more important for software engineering. The author hopes he has provided the students a good framework for exploring this important and profound issue in greater depth as they advance in their careers.

8. References