Proceedings of the 11th Colloquium for Information Systems Security Education
Boston University
Boston, MA June 4-7, 2007

CAN SOFTWARE ENGINEERS BE BOTH AGILE AND SECURE?

Richard G. Epstein, *West Chester University of Pennsylvania*, Member IEEE, ACM

Abstract – This paper describes an undergraduate course in software engineering that introduces students to a variety of processes that are used to develop software. Students are asked to consider the security implications of the various processes. Special emphasis is given to PSP, CMM and agile processes (like eXtreme Programming and Scrum). An important issue in this course is whether agile processes can produce secure software and, if not, how they might be improved to make agile processes more secure. Students work on a major team project that involves developing a software process for a pretend company and a team presentation project that addresses the security issues specifically.

Index terms – Software processes, building secure software, ethics, professional responsibilities[°]

1 – BASIC GOALS OF THIS COURSE

The author has designed an undergraduate software engineering course for majors in Computer Science that introduces them to a variety of software processes and the security implications of those processes. The course illustrates that there are many different software processes "out there" and that each individual organization needs to develop a process (or, a set of processes) that is appropriate for its development environment (or, set of environments). The course introduces the students to what has been called "the war" between the traditional waterfall processes and the agile processes. One aspect of this war is that agile processes have received some bad press with regard to their ability to produce secure software. The author believes that this focus on the conflict between the traditional waterfall process folks and the agile process folks adds quite a bit of energy to the course. The students are placed in a situation in which they must "choose sides," and determine the strengths and weaknesses of the various processes, especially in terms of their ability to produce secure software. Some students seem especially interested in tackling the problem of how agile processes can be made more effective in terms of their ability to produce software with fewer vulnerabilities.

The basic goals in this course include: (1) Introducing students to a variety of software processes, (2) Giving students the opportunity to work in teams to design a defined software process for a pretend company, (3) Introducing students to ethical and work culture issues in software engineering, (4) Introducing students to quality assurance measures in the software development process, and (5) Introducing students to security issues in software engineering. Indeed, the security issues are interwoven throughout the course. This paper will describe how security concerns do get integrated into most of the material covered in the course, including our discussions relating to software processes, work culture issues, the professional and ethical responsibilities of software engineering professionals, and quality assurance measures in the software development process.

Although this paper will focus on the undergraduate version of our university's software engineering course, this paper will, in one specific context, refer to the graduate version of the course. An important aspect of the graduate version of the course is that each student is required to write an individual research paper. Although the graduate students were able to choose from a variety of topics for their research papers, the relationship between agile processes and security was a topic that seemed to stimulate a lot of interest and several papers of high quality were submitted relating to this thought-provoking subject.

The course begins with an introduction to a variety of software processes. This is the device that the author uses to introduce the students to "the war" mentioned above. The course begins with an introduction to a traditional waterfall process, the Personal Software Process from Watts Humphrey at the Software Engineering Institute. The course uses the Personal Software Process as a device for introducing students to a traditional waterfall process model that requires a lot of data collection. The course also provides students with an introduction to the Software Engineering Institute's Capability Maturity Model Integrated (CMMI), which while not a software process, is an important model for evaluating the effectiveness (or, "maturity") of an organization's

[°] *Richard G. Epstein, Department of Computer Science, West Chester University of Pennsylvania, West Chester, PA, 19383.*
Web site: www.cs.wcupa.edu/epstein;
Email: repstein@wcupa.edu

software process. Usually, the CMMI is associated with traditional waterfall software processes, although that is not a requirement within the model.

The course then introduces the students to "the war" by introducing them to agile processes, with an emphasis on eXtreme Programming and Scrum. This leads to some animated classroom discussion concerning the strong points and weak points for the various processes. The author definitely gets the sense that the majority of the students are inclined to favor the agile processes over the traditional waterfall processes. Agile processes seem to be "more fun" and industry research indicates that developers tend to enjoy working in an agile environment. In other words, agile processes may improve workplace satisfaction.

While security issues are introduced with respect to some of the CMMI process areas, it is after our discussion of agile processes that security issues are introduced more formally. In particular, students are introduced to the view of some experts in computer security (like John Viega, Gary McGraw and Eugene Spafford) that agile processes might not be consistent with the goal of developing secure software. The students are given a general introduction to security issues in software engineering and considerable attention is given to how agile processes (with a focus on eXtreme Programming, in particular) might be improved in terms of producing secure software.

The following sections of this paper give more details regarding the topics covered, the resources that are used to introduce security concerns into the course, the nature of the team projects that are assigned to the students, and some of the creative suggestions made by the students relating to how agile processes might be made more secure.

2- SOFTWARE PROCESSES

The focus at the beginning of the course is on software processes. We begin with a discussion of Fred Brooks' classic paper "No Silver Bullet" [1]. It seems that twenty years after Fred Brooks wrote this influential paper, there is still no "silver bullet" to slay the software engineering werewolf. 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 that comes from the traditional waterfall end of the spectrum. Although there is no team component in PSP (as opposed to the 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.

Later in the course we discuss a report from a committee formed by the Department of Homeland Security (the US National Cybersecurity Summit Subgroup), a report which makes recommendations in terms of what the committee believed are good processes for developing secure software (Davis et al. [4]). This report introduces students to an extended version of TSP that covers security issues (TSP-Secure). The report from the Cybersecurity Summit Subgroup makes it clear that processes that produce low defect densities (a metric introduced in PSP and TSP) are likely to be the most secure. The DHS Cybersecurity Summit Subgroup report (as published in IEEE Security and Privacy) does not mention a single agile process.

The course then moves on to agile processes, with an emphasis on eXtreme Programming (XP) (for example, see the book by Beck [5]) and Scrum (for example, see the book by Schwaber [6]). Agile processes represent a 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, project planning (achieved by prioritizing the user stories), iterative development, emphasis on simplicity, code refactoring, pair programming, writing the test code before the product code is written, continuous integration, acceptance testing, and the forty hour work week. The conventional discussions of XP do not focus on security issues at all. So, the students are alerted early on that security issues are going to be a major concern for us as we consider the appropriateness of agile processes, like XP, for developing secure software. We also spend time discussing Scrum, which is closely related to XP, and to meta-Scrum, which addresses the issue of scaling up agile processes to larger projects. We also discuss how XP practices have been scaled up to handle larger projects (as reported in the paper by Talby et al. [7]).

XP exposes students to a recurring theme in the course: four eyeballs are better than two in order to prevent defects from being introduced into the delivered product. This eventually evolves into the "many eyeballs principle" for quality assurance. We also discuss XP from an industry perspective, discussing some papers that

discuss the use of XP and Scrum in industry (e.g., the excellent papers by Grenning [8] and Schatz et al. [9]).

Next, we introduce the Capability Maturity Model (CMM and, more recently, CMMI) from SEI. We introduce CMM using a provocative paper by Mark Paulk entitled "Extreme Programming from a CMM Perspective" [10]. Paulk's paper accomplishes several things in terms of the goals of this course. First of all, it introduces the CMM and basic CMM terminology (e.g., levels of maturity and key process areas). Secondly, 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 XP framework. (We supplement this material from Mark Paulk with more recent materials relating to the process areas in the new Capability Maturity Model Integrated, CMMI.)

Mark Paulk was involved in the development of the CMM in the early 1990s at SEI. Paulk shows that XP is not inconsistent with CMM. He explores the strengths and weaknesses of XP in terms the CMM key process areas. We find out that an organization that uses XP could certainly achieve level 2 in the CMM framework and could satisfy some of the key process areas for level 3 (and even one at level 5, the highest CMM maturity level).

It is well known within the agile processes community that Mark Paulk is quite unhappy with some aspects of the CMM framework. (Again, Paulk was involved in the original development of CMM at SEI.) More precisely, Paulk has expressed some ambivalence concerning the impact CMM has had on development organizations, including the impact it is having on workplace satisfaction. Ken Schwaber was one of the folks who developed the Scrum agile process. On a Scrum Web site there is an audio available in which Schwaber, describes a panel discussion which was set up at a conference in order to allow "the war" to manifest clearly. Mark Paulk was on the traditional waterfall side of the panel (associated with CMM) and the other half of the panel was filled with agile process folks, including Schwaber. Much to everyone's surprise, Mark Paulk attacked some aspects of CMM (or, at least, the impact it was having on the software development community in terms of productivity and work enjoyment) and stressed that the waterfall folks had a lot to learn from the agile folks. More recently, there has even been talk of the need to develop a maturity model for agile processes, a model that would include consideration of security issues. It was interesting to see how excited the students in the class became when they heard about this proposal: an Agile Process Maturity Model!

It is interesting to note that almost none of the resources that we use to introduce students to software processes devotes much attention to security concerns right up front. The author tries to alert students to these issues early on, in an iterative fashion.

3 – PROFESSIONAL AND ETHICAL ISSUES

Once the students have been introduced to several important software processes, we shift our focus 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. First we cover the Therac-25 accidents that involved very subtle programming errors that caused a radiation therapy machine to kill three patients and injure three others. These accidents are discussed in a powerful paper by Leveson and Turner [11]. We then discuss the Confirm fiasco, in which an integrated airline, hotel reservation, and rental car reservation system was never completed because of the incompetence and one might argue the unethical behavior of some of the project leaders (see Oz [12]).

The discussion of the Therac-25 and Confirm fiascos naturally leads to a discussion of the Software Engineering Code of Ethics (Gotterbarn et al. [13]). 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.

An important ethical concern for software developers is to produce software that is secure. The three papers alluded to in the preceding paragraphs in this section (including the Software Engineering Code of Ethics) do not emphasize security as an ethical concern. If one makes an effort, security issues can be read into the subtext of the Software Engineering Code of Ethics. Security is a fundamental concern when one discusses issues such as the reliability and robustness of the systems software engineers are creating. Also, security is clearly in the subtext when the Software Engineering Code of Ethics stresses the importance of protecting users and society from harm. Clearly, there is much more focus on security as an ethical issue in software engineering now than there was ten years ago.

4 – WORK CULTURE AND QUALITY ASSURANCE ISSUES

The course then moves on to discuss a variety of issues in software development, including work culture issues and

quality assurance. The discussion of work culture focuses on the idea of "congruence" presented by McLendon and Weinberg [14]. McLendon and Weinberg 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 (that is, an incongruent 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 (Epstein [15]) that can have a negative impact upon the work culture. Students participate in a class exercise that 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" [16]. 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 detrimental (or, counterproductive) state of mind.

When it comes to quality assurance, we look at industry-based studies of quality assurance practices, including software reviews and software testing. One interesting article that we discuss looks at the conflicts between software developers and software testers (Cohen et al. [17]). This article combines the work culture issues discussed earlier and relates those issues to fundamental quality assurance practices.

Clearly, an important aspect of quality assurance is assuring that the software is secure. Our discussion of team culture issues (and the conflicts between testers and developers as described by Cohen and her co-authors) gives us an opportunity to discuss the role of security personnel in the software development process. For example: What are the conflicts that might exist between a security expert on a software development team and the other software developers? We examine software project goals (like user-friendliness) and security goals (like authentication) and how different constituencies (like security experts on the one hand and developers on the other) might differ on ranking the importance of conflicting goals. More generally, we consider how different constituencies involved in a software project (like the client, the security experts, the software developers, the maintainers, the users) may have conflicting perspectives when it comes to discussing the relative importance of software project goals and security goals.

Indeed, these kinds of tensions will be one of the major foci in the next portion of the course (discussed in more detail in the next section). The students were being introduced step by step (i.e., in an iterative fashion) to the concept that security concerns need to be integrated into the software development process from the beginning, whether one is using a traditional waterfall process or an agile process. Now, the course focuses on security as a fundamental concern that needs to be integrated throughout the software development process.

5 – ISSUES IN SECURITY

The course tries to alert students to the security issues in software development from day one. Occasionally, as we discussed software processes, work culture issues, ethical and professional responsibilities, and quality assurance, we discussed the security concerns that are now central to software engineering. This next component of the course is devoted to a detailed discussion of security issues specifically.

This portion of the course uses a high-level approach to security issues, rather than focusing on specific coding issues for specific languages. This high-level approach involves emphasizing how security concerns should be integrated into the software development process during each phase of the process. We begin the discussion with a general overview of security goals and principles. We draw heavily upon the principles enunciated in the aforementioned report from the Department of Homeland Security (Davis et al. [4]) and the excellent book by Viega and McGraw [18]. These basic principles include defense in depth, the principle of least privilege, simplicity, using community resources, and being reluctant to trust. These principles can be at odds (as mentioned above) with basic software project goals, and they can also be in conflict with one another. In addition, different constituencies will have different perspectives on the security goals and the software project goals.

We discuss how security can be incorporated into the requirements gathering, design, implementation, and testing phases of the project. An important topic in this portion of the course is how risk assessment needs to be used in order to determine the eventual functionality of the system. Other topics that are covered in these lectures about security include security testing, the use of scanning tools to detect security vulnerabilities, and the controversy concerning the security or insecurity of Open Source code.

The course then moves on to assess software processes in terms of their effectiveness in terms of producing secure software. We begin this discussion with the article by Noopur Davis et al. [4] (this is the report by the Department of Homeland Security Cybersecurity Summit Subgroup alluded to earlier). The Cybersecurity Summit Subgroup report supports certain processes as being

consistent with high quality and secure software. Agile processes are not among the recommended processes in this report. We then turn our discussion to eXtreme Programming and the concerns that some authors have expressed regarding its appropriateness for developing secure software. For example, in their book, Viega and McGraw [18] devote just one paragraph to XP, voicing their opinion that it is not a good process for developing secure software. Our goal in this section of the course is to have an animated discussion about how the basic practices of XP could be modified to explicitly incorporate security concerns into the XP framework.

The graduate version of the course requires that students write an individual research paper. Several of the papers were of exceptional quality, and one student (in particular) wrote a very interesting research paper that is very relevant to the matter at hand (i.e., can agile processes be secure?). This student's paper presented the framework for a new software process that he called XP-Secure. There were many interesting ideas in this paper. One idea in particular related to how XP-Secure would modify its requirements gathering process. Requirements gathering in XP-Secure would involve not only the collection of user stories (brief descriptions of system functionality collected from the client) but also security stories, relating to the security requirements for the system. The security stories would be written primarily by a security expert on the team, who would interact with the developers and the clients. This student's paper then went on to discuss how the security stories would influence the planning process and the iterative development of the final system. Thus, the final system would be designed to make BOTH the clients AND the security experts happy. XP-Secure also showed how security concerns could be integrated into XP's basic practices, such as pair programming, continuous integration, daily stand-up meetings and so forth.

An important session in both the undergraduate and graduate versions of this course involves an in-class discussion of two issues:

- Possible tensions / conflicts between the basic principles and goals of secure software development and the basic practices of XP, and
- Ways in which the basic practices of XP can be modified to address those principles and goals.

The author gives the students a handout which presents a matrix. Running down the left-hand side of the matrix are basic security principles and goals extracted from Viega and McGraw's book [18] and the DHS article [4]. The specific principles and goals chosen for this discussion are:

- Prevent vulnerabilities that can lead to exploits
- Implement auditing and / or monitoring
- Ensure confidentiality, integrity and accessibility
- Provide multilevel security
- Provide trustworthy authentication processes
- Follow the principle of least privilege
- Fail securely
- Be reluctant to trust
- Use community resources

Running across the top of the matrix are the basic practices of XP, namely:

- Customer on-site at all times
- Requirements gathering using user stories
- Project planning (including release planning)
- No big up-front design
- Pair programming
- Unit tests written before actual code
- Acceptance testing guided by customer
- Frequent integration of new components into code repository
- Daily stand-up meetings
- Do not add functionality the customer does not want
- No overtime! Forty hour work week
- Redesign code to keep it simple and correct (refactoring)

The discussion of how the XP practices can be modified to accommodate security concerns is very stimulating. The discussion usually centers around the idea that the software development team should include a security engineer. The security engineer can help modify the XP basic practices to satisfy the security principles and goals. Recommendations include the use of static scanners, security stories (as recommended by the graduate student mentioned above), security-oriented design (beyond just the usual CRC cards, perhaps using a notation like UMLsec, as described in Apvrille et al. [19]), security-oriented testing, and so forth.

The author can only conclude, on the basis of this kind of discussion, that agile processes do not intrinsically contradict the goal of developing secure software. What is needed at out there in the software development community is a lot of exploration and experimentation with upgraded versions of "traditional" agile processes, versions that include security concerns right from the beginning. The author is convinced (and his students helped him to come to this conviction) that agile processes that are consistent with the goal of producing secure software can be developed. (In other words, software engineers can be both agile and secure!)

This portion of the course includes a discussion of how security practices have been incorporated into software projects in industry. A good resource for this discussion is the article by Apvrille et al. [19] which describes an experimental project which attempted to show how

security issues could be integrated into the software life cycle right from the beginning (much in the style recommended by Viega and McGraw [18]).

6 – TEAM PROJECTS

The undergraduate version of the course has the students work on two team projects. The first of these projects is to develop a defined software process for a pretend company. It is due about two-thirds of the way through the course. Students are asked to integrate ideas from the various processes that we discuss in order to create a defined software process for their pretend company. The final team project, delivered at the end of the semester, involves an in-class team presentation that is intended to synthesize ideas from the entire course, including the concerns relating to building secure software. The students are asked to pay special attention to the security concerns in their final presentations.

The emphasis for these in-class team presentations is on creativity. While some teams end up giving a PowerPoint presentation (and many 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 (with a video camera) 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 truly enjoys teaching this software engineering course. He has gotten excellent feedback from students, including comments from some alumni that the manner in which the course covered software processes turned out to be very helpful for them when they went out 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 CITED

1. Brooks, Fred, "No Silver Bullet", IEEE Computer, April 1987, pp. 10-19.

2. Humphrey, Watts S., Introduction to the Personal Software Process, Addison Wesley, Boston, 1996, 304 pp.

3. Morisio, Maurizio, "Applying the PSP in Industry," IEEE Software, November/December 2000, pp. 90-95.

4. Davis, Noopur, Humphrey, Watts, Redwine, Samuel T., Zibulski, Gerlinde, and McGraw, Gary, "Processes for Producing Secure Software", IEEE Security and Privacy, May/June 2004, pp. 18-25.

5. Beck, Kent, eXtreme Programming eXplained: Embrace Change, Addison Wesley, Boston, 2000, 190 pp.

6. Schwaber, Ken, Agile Project Management with Scrum, Microsoft Press, Redmond, WA, 2004, 192 pp.

7. Talby, David, Kren, Arie, Hazzan, Orit, and Dubinsky, Yael, "Agile Software Testing in a Large-Scale Project," IEEE Software, July/August 2006, pp. 30-37.

8. Grenning, James, "Launching eXtreme Programming at a Process-Intensive Company," IEEE Software, November / December 2001, pp. 27-33.

9. Schatz, Bob, and Abdelshafi, Ibrahim, "Primavera Gets Agile: A Successful Transition to Agile Development," IEEE Software, May / June 2005, pp. 36-42.

10. Paulk, Mark C., "Extreme Programming from a CMM Perspective", IEEE Software, November / December 2001, pp. 19-26.

11. Leveson, Nancy G., and Turner, Clark S., "An Investigation of the Therac-25 Accidents", IEEE Computer, July 1993, pp. 18-41.

12. Oz, Effy, "When Professional Standards are Lax", Communications of the ACM, October 1994, pp. 29-36.

13. Gotterbarn, Don, Miller, Keith and Rogerson, Simon, "Software Engineering Code of Ethics", Communications of the ACM, November 1997, pp. 110-116.

14. McLendon, Jean and Weinberg, Gerald M., "Beyond Blaming: Congruence in Large Systems Development Projects", IEEE Software, July 1996, pp. 33-42.

15. Epstein, Richard G., "Demons in the IT Workplace", IEEE ISTAS, Worcester, MA, June 2004.

16. McConnell, Steve, "Problem Programmers," IEEE Software, March/April 1998, pp. 128-7.

17. Cohen, Cynthia F., Birkin, Stanley J., Garfield, Monica J., Webb, Harold W., "Managing Conflict in Software Testing," Communications of the ACM, January 2004, pp. 76-81.

18. Viega, John and McGraw, Gary, Building Secure Software, Addison-Wesley, Boston, 2002, 493 pp.

19. Apvrille, Axelle and Pourzandi, Mahan, IEEE Security and Privacy, July / August 2005, pp. 10-17.