Getting Students to Think About
How Agile Processes Can Be Made More Secure

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

Abstract

Agile processes play an important role in the author’s undergraduate course in software engineering. The course is a required course for undergraduate majors in Computer Science. Agile processes, like eXtreme Programming (XP), have been criticized for not providing a good framework for building secure software. The course begins by covering what some people have called “the war” between the traditional waterfall process folks and the agile process folks. After students are given an introduction to various processes on both sides of “the war” (with an emphasis on PSP, CMMI and XP) and after students are introduced to basic concepts about how to make software systems more secure (drawing heavily on Viega and McGraw’s book Building Secure Software), the course turns its attention to how XP (in particular) can be made more secure. This topic generates a lot of enthusiasm among the students. The students seem to enjoy the challenge of creating new ideas to improve the manner in which XP addresses security issues. Students have come up with many creative and stimulating ideas about how eXtreme Programming can be made more secure without the necessity for what some have called “big up front design”. This paper presents some of the creative ideas students have come up with regarding this issue and discusses the team projects that give students the opportunity to explore security issues for agile processes in some depth.

1. Introduction

Around the turn of the century (2000, not 1900), the author began to include eXtreme Programming [1,2] in his undergraduate course in software engineering. Two basic goals of this course were to introduce students to a variety of software processes and to get students to think about the strengths and weaknesses of the various processes that are discussed. However, within a year or two after he introduced eXtreme Programming into his software engineering course, the author realized that many experts have reservations about eXtreme Programming in terms of its appropriateness for building secure software.

The author came to believe that eXtreme Programming (XP) could be made more secure and that getting students to think about how this might be done would make a positive contribution to the course content. XP does seem to have some shortfalls in terms of producing secure software. The course attempts to get the students to think about these shortfalls and also to think about ways to improve XP so that the shortfalls might be overcome. The author believes that this is an interesting problem and he has seen that it generates a lot of interest in our classroom discussion, as we shall see later in this paper.

First, this paper gives an overview of the course, in order to provide the context in which this discussion takes place. The paper then turns its attention to how security issues are introduced into the course. The main focus shall be on how the course gets
students to think about how XP might be made more secure. Specific suggestions that arise during our in-class discussion will be presented.

2. Overview of the course

The course begins by introducing students to a variety of software processes. We begin with a two-week introduction to PSP from Watts Humphrey at the Software Engineering Institute [3]. This is followed by an introduction to agile processes, with an emphasis on XP and Scrum.

The course relies on an online “coursepack” which consists of a collection of articles from the ACM and IEEE Computer Society digital libraries. Indeed, the digital libraries are an invaluable resource for this course. For example, our introduction to PSP, XP and Scrum includes discussions regarding how these processes have been applied in industry as reported in several articles from the digital libraries. Morisio [4] tells us about his company’s attempt to incorporate PSP into the work environment. Although most of the developers abandoned most of the PSP practices, when given the choice to do so, they were impressed by PSP’s quality assurance measures, specifically defect tracking. Grenning [5] describes how he led a project that introduced XP into his company’s software development culture. Grenning makes the important point that XP needed to be adapted to the corporate environment. This is an important point because there is more and more flexibility in the XP and agile process software development culture. This makes it easier to modify agile processes in order to make them more secure. Schatz and Abdelshafi [6] discuss how Scrum was successfully introduced into their corporate environment with the help of their Scrum mentor, Ken Schwaber.

Because of time limitations, we discuss TSP only briefly, and then move on to discuss the SET’s CMM (now, the CMMI) framework for evaluating process maturity. An excellent paper by Mark Pauk [7] allows us to integrate the CMM concepts into our earlier discussion of XP. We learn that XP is not inconsistent with the framework provided by CMM. We conclude our discussion of software processes with a brief introduction to Open Source software development.

After completing our discussion of software processes, we move on to professional issues for software engineers. We discuss well-known software disasters (like the Thrac-25 accidents, discussed by Leveson et al. [8] and why software fails (using the framework provided by Charette [9]). This naturally leads into a discussion of the Software Engineering Code of Ethics [10]. Clearly, one of the ethical obligations of a software engineer is to make an effort to provide software that is secure (or, perhaps we should say, reasonably secure).

The course then moves on to discuss work culture issues in software engineering with an emphasis on the ideas of McLendon and Weinberg [11] relating to incongruent work cultures and blaming in the software development workplace. The course then moves on to a discussion of quality assurance issues. An important paper that we use in the quality assurance section of the course is Talby et al. [12]. Talby and his co-authors describe how XP was scaled up for a large project which had serious security and reliability considerations. Although the authors do not focus on security per se, they do make it clear that software testing was done in this XP project in a rigorous and intense manner.

The article by Talby et al. provides a good segue into a discussion of how security issues can be integrated into a software process, which then leads into our discussion of how agile processes (XP, in particular) can be made more secure. However, before moving on to how these issues are treated in the course (in Sections 4, 5 and 6), let’s put in a few words about “the War”.
3. “The War”

The diverse software processes that are discussed in this course are central issues in what some software engineers are calling “the War”. The War is the conflict between the hard-core data collection and documentation folks on one side and the agile process folks on the other. The author's recollection is that he first heard the phrase “the war” in a conference keynote address given by Ken Schwaber, a key person behind the development of the Scrum agile process [13]. Schwaber describes a panel discussion at another conference in which there were two tables set up on the stage. The two tables were set up to reflect the two sides in the so-called “War”. At one table you had people who were supporting CMM (now called CMMI), including Mark Pauk. On the other side you had people who were supporting agile processes, including Ken Schwaber.

The CMM folks did not like the agile approach to software development and the agile process folks did not like the CMM approach. That’s the War. This author likes to say that the CMM folks accuse the agile folks of producing WMDs (Web-apps of Mass Disruption) while the agile folks accuse the CMM folks of producing WMDs (Worthless Massive Documents). Indeed, part of this war is the perception on the CMM side of the argument that the agile process folks do not do enough big-up-front design. There are serious security implications, according to the CMM folks, for the lack of big-up-front design among the agile process folks. The agile process folks, on the other hand, emphasize iterative development. They claim that the CMM just results in massive documents that no one ever reads.

In Ken Schwaber’s description of this panel discussion, he describes an important departure from the expected. Mark Pauk, sitting at the CMM table, spoke up in defense of the agile processes, claiming that there was a lot that the CMM folks could learn from the agile process folks. He criticized many CMM projects for the great burden they placed on developers and indicated that workplace dissatisfaction was one of the consequences of the need to create massive documents that hardly anyone would ever read. The arguments and counterarguments relating to the “War” are discussed in several web sites, including [14]. The war also manifested in a Point-Counterpoint article in IEEE Software [15]. The main focus in the Point-Counterpoint article is whether you really need big-up-front specifications in a software project.

One aspect of “the War” is whether agile processes are appropriate for producing secure software. In their excellent book Building Secure Software [16], Viega and McGraw only mention eXtreme Programming briefly and only once and this is what they had to say: “In general, the cleaner and easier the specification, the better the resulting system. Unfortunately, extreme programming (XP) seems to run counter to this old-fashioned view! In our opinion, XP will probably have a negative impact on software security [16, p. 35].”

4. Introducing security into the course

Students are introduced to “the War” early on in the course. As suggested at the end of the previous section, an important point of contention in this War is whether agile processes are appropriate for building secure software. Before we can explore this issue in depth, we need to look at what various experts have to say about incorporating security concerns into the software development process.

Viega and McGraw [16] provide important concepts for this portion of the course. In addition, we draw upon a report from the Security Across the Software Development Lifecycle Task Force that was organized at the National CyberSecurity Summit in December 2003. Members of the Task Force included representatives from industry, academia and the United States Department of Homeland Security. The report, as summarized in Davis et al. [17], identifies good processes for building secure software. It is important to note that agile processes are not mentioned at all in this paper. The paper focuses on good processes for
developing secure software, as opposed to explicitly criticizing processes that they felt were not so good. It is interesting that both Watts Humphrey and Gary McGraw were on this Task Force. Among the processes that the committee recommends are TSP-Secure, Correctness-by-Construction, and Cleanroom Software Engineering. In addition, they recommend the CMMI framework (specifically, SSE-CMM, or Systems Security Engineering-CMM). An important point that the members of the Task Force make in their paper is that there is a correlation between low defect densities in the code and good security.

Viega and McGraw (in [16]) provide an excellent framework for discussing how security issues can be integrated into the software development process. They discuss the importance of developing a security policy early on, in the requirements analysis phase. They discuss how risk assessment needs to be integrated into the requirements analysis phase, so that a complete risk assessment is completed before the project can move on to design and implementation. They also discuss trust assumptions, and their importance for secure design, and the importance of using appropriate tools to check software for security vulnerabilities. They discuss the need to train software developers in secure coding practices. They also discuss security testing, and how it differs from normal software testing. In particular, security testing focuses on what the “bad guys” might do to discover vulnerabilities in the software.

Our class discussion then moves on to a paper that illustrates how the a group of researchers integrated security concerns into a software development project very much in the style suggested by Viega and McGraw (Apvrille et al. [18]). This project involved developing an instant messaging system that the authors called PICO. They describe how security concerns were factored into each phase of the software life cycle, much in the manner that Viega and McGraw recommend.

Viega and McGraw present goals and principles for secure software and they discuss the conflicts that might exist among these goals. In addition, they discuss how the security goals might be in conflict with the software project goals. For example, the security goal of “defense in depth” can be at odds with the software project goals of timeliness and user-friendliness. This discussion of conflicts among the various project goals (whether generic project goals or security goals) naturally leads into our next topic: agile processes and security.

5. Agile processes and security: the Matrix

The author does his best to get students to see the issue of agile processes and security as an interesting one. In particular, looking at this issue gives us an opportunity to be creative in the sense that we make an effort to describe how an agile process, like eXtreme Programming, can be modified so as to make it more appropriate from a security perspective. By the time we get to the class that is devoted specifically to this topic, many students do seem to be generally interested in this issue and try to contribute creative ideas to our in-class discussion.

The initial discussion centers around a handout that the author calls “the Matrix”. The Matrix has nothing to do with the movie by that title. Rather, it is a table whose rows are labeled (on the left) with basic principles for building secure software (drawn from Viega and McGraw [16] and Noopur et al. [17]). The columns are labeled (across the top) with the basic practices of eXtreme Programming. Students are asked to focus on the intersections between the rows and the columns in this table (or, matrix) and to comment either on the tensions between the security principles and the XP practices or the ways in which the principles and practices might reinforce one another. For example, one of the security principles listed on the
left is “Provide multi-level security”. One of the XP practices listed on the top of the matrix is “Do not add functionality the customer does not want”. Students are asked to consider ways in which the security principle might be in conflict with the XP practice.

Here is the complete list of the security principles listed on the left-hand side of the matrix:

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

Here is the complete list of the XP practices that are listed across the top of the matrix:

- Customer on-side at all times
- Requirements gathering using user stories
- Project planning (release planning, iteration 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 goal is to get students to see the possible conflicts and tensions between the security principles and goals and the XP practices. In addition, students might see situations in which the security principles and XP practices are in synch. The author has found that students sometimes have difficulty seeing these relationships. They seem to get more “turned on” when the discussion turns to suggesting ways in which XP can be improved in terms of security.

6. Some recommendations for making XP more secure

The students have come up with some interesting ideas for making XP more secure. Most of these ideas come up during our in-class discussion of XP and security. Others have found
their way into the team projects that students submit at the end of the semester and individual research papers students submit at the end of the graduate version of the course. In fact, one graduate student wrote a research paper about a process he called XP-Secure. Although the following discussion is not based on the ideas in his paper, we will refer to our secure version of XP using his terminology: XP-Secure. (It should be noted that some of the ideas that follow, were in his paper, specifically the idea of security stories. However, quite a few students seem to come up with similar ideas in our undergraduate class discussion of how XP can be made more secure.) The following paragraphs describe some of the ideas that emerge during our in-class discussion of how to make XP more secure.

One key idea is the need for a security engineer on the development team. Just as XP requires that a customer be on site all the time as part of the development team, the students saw the need for a security expert on the XP-Secure development team. The security engineer would play an important role in guiding the team's effort in the direction or producing a secure product. The security engineer would interact both with the developers and with the on-site customer. One goal of the security engineer would be to educate both the developers and the customer about how the project would be impacted by security issues and concerns. This kind of input from the security engineer would occur during all of the phases of the project.

Another interesting idea that students have suggested is the need for security stories to supplement the user stories that XP uses to determine the requirements. The evolving collection of security stories and user stories would become the basic requirements framework for the project. Security stories might come directly from the security engineer or they might be developed by the customer as a result of the customer's interaction with the security engineer. The security engineer could see the implicit security issues involved in the functionality that the customer describes and thus the security engineer could develop security stories as a way of specifying the project's security policy. In other words, security stories would play a major role in determining the security policy for the project.

A big issue for XP-Secure is how project planning would be conducted. Here is the basic framework that emerges from our in-class discussion of project planning for XP-Secure. First of all, user stories and security stories would need to be prioritized, which is the usual practice at the release planning meeting. However, risk assessment must be part of this process of prioritizing the user stories and the security stories. The security stories that are agreed upon, during the release planning meeting with the appropriate risk assessment having been performed, will constitute the security policy for the project, just as the user stories will define the functionality of the project.

One important issue that gets raised at this point in the discussion is whether the security policy can be decided upon “up front” at the release planning meeting. How will this differ from a more traditional “big up front design” process? Also, will the security policy make demands upon the developers that will make the traditional XP design process inappropriate? Will larger architectural issues need to be discussed right “up front” in the XP-Secure process, or can these decisions be made later on, as in the usual XP process?

Let's assume that XP-Secure is highly iterative. The release planning meeting gives rise to a project plan laid out in terms of tentative iterations. At the end of each iteration, the project must satisfy the basic security goals stipulated by the security stories for that iteration and the security policy considerations that emerge from those security stories. A security story that is not satisfied at the end of an iteration must be addressed in the subsequent iteration. Any software released to the customer must satisfy the security policy that the security engineer
helped to specify with the original collection of security stories agreed upon at the release planning meeting.

Here are some of the questions that often come up in discussing the design aspects of XP-Secure: Are CRC (Class Responsibility and Collaboration) cards sufficient for designing a secure architecture? One possibility is to bring UMLsec into the project, as described in the afore-mentioned paper by Aprille et al. [18]. Students wonder whether XP allows for adequate attention being paid to issues in software architecture that relate to trust assumptions and relationships between the software components. They question whether a highly iterative process can handle these problems adequately.

In terms of implementation and continuous integration, here are some of the points that arise during our discussion of XP-Secure. First of all, the security engineer must be involved in the code review process. The security engineer should be involved in training the developers in terms of secure coding practices so that the quality assurance benefits of pair programming will carry over into the realm of writing secure code. Unit testing must be expanded to include static testing tools and also the new dynamic testing tools that are coming out into the marketplace. In other words, testing must be expanded to include security testing explicitly. Security testing must also be integrated into the XP continuous integration process. Developers must be trained explicitly to do security testing and to integrate the results of the security tests into the iterative process, including the daily stand-up meetings. Acceptance testing must be guided by the security engineer as well as by the customer.

Here are some additional points relating to how XP-Secure might differ from XP. The daily stand-up meetings must include input from the security engineer, including reports on how the security-oriented tests are going. The security test results (whether involving static testing or dynamic testing, whether unit tests or integration tests or acceptance tests) may have a big impact upon the tasks that are assigned for a particular day. Another point that is made is that all refactored code must be analyzed and tested from a security perspective. Any user story or security story that has not been successfully tested from a security perspective cannot be considered complete and cannot be included in a product release.

7. Conclusions

Agile processes are increasingly important in the world of software development, especially in this age of Net-oriented software. An important issue is how agile processes, like eXtreme Programming and Scrum, can be modified to address important security issues. This paper looked at how the author gets students in his software engineering class to consider these issues and how he encourages them to see ways in which agile processes might be made more secure.

8. References


