Proceedings of the 13th Colloquium for Information Systems Security Education
University of Alaska, Fairbanks
Seattle, WA June 1 - 3, 2009

INTEGRATING SECURITY INTO AN UNDERGRADUATE COURSE ON SOFTWARE EENGINEERING

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

Abstract – ***This paper describes an undergraduate course in software engineering which introduces students to a variety of approaches to developing software. These include PSP, CMMI and agile processes, such as XP and Scrum. An important element in the course is getting students to consider how security issues arise during the software development process. Security issues are raised with respect to the software processes themselves, as well as in our discussions of professional responsibilities, ethics, work culture issues and quality assurance.***

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

I. BASIC GOALS OF THIS COURSE

This paper presents an overview of an undergraduate software engineering course for Computer Science majors that introduces students to various processes for developing software. This course is taught during the senior year. The basic prerequisite for this course is completion of a four course sequence that introduces students to programming, data structures, and the analysis of algorithms. The course introduces the students to ideas relating to how security concerns can be integrated into various software processes. The course emphasizes the fact that each organization should try to develop processes that are appropriate for its development environments. There is no single process that will work in all situations and in all contexts. In addition, the course introduces students to the idea that software processes need to evolve as organizations learn from their successes and from their mistakes. The basic goals in this course include:

- Introducing students to a variety of software processes that are used in industry,
- Giving students the opportunity to work in teams to develop a software process for an imaginary company,
- Introducing students to security issues in software engineering,
- Introducing students to professional, ethical, and work culture in software engineering, and
- Introducing students to quality assurance measures in software engineering.

Following our introduction to various software processes, including the Personal Software Process (PSP) and eXtreme Programming (XP), we turn our attention to how security concerns need to be integrated into the software development process. This, in turns, leads to an animated discussion of how agile processes (and XP in particular) might be modified so as to make them more compatible with the goal of building secure software. Various authors have criticized XP on the grounds that it is not a good process for producing secure software. Over the past few years, students in this software engineering course have come up with some interesting ideas about how XP can be modified to address these security concerns.

The following sections give more details regarding the topics covered, the resources that are used to introduce security concerns into the course, the discussion as to how agile processes might be modified to accommodate security concerns and the nature of the two team projects that students work on in this course.

Please note that almost all of the references cited at the end of this paper are in what the author calls " the coursepack" for this course. The coursepack is not a printed booklet. Students have access to these articles online through the ACM and IEEE Computer Society digital libraries. The coursepack articles, the author's lecture notes and PowerPoint slides (available to the students via Blackboard) and the Watts Humphrey book on the Personal Software Process all play a key role in communicating the ideas in this course.

II. SOFTWARE PROCESSES

The focus at the beginning of the course is on software processes. We begin with a general introduction to software engineering (with the lecture notes made available to the students via Blackboard) 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 introduction to PSP is to give students a concrete example of a well-defined and rigorous software development process. Although there is no team component in PSP (as opposed to the Team Software Process, TSP, which we also discuss), PSP provides the students with a basic introduction to important ideas in software engineering. These include project planning, collecting data on time

spent on various activities, and defect tracking. Humphrey's ideas regarding quality assurance are 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 (in terms of the willingness of developers to stick with the data collection regimen, as discussed in [3]), it is a good way to introduce students to what team-oriented processes like TSP (which builds upon PSP) entail. We discuss briefly an experiment at Microsoft in which development teams used TSP.

The author introduces the students to Watts Humphrey by reading an excerpt from a book by David Rice that bears the title Geekonomics: The Real Cost of Insecure Software [4]. Rice's message in this book is an important one for the course: we are using insecure software to build our critical information infrastructure and this can have devastating consequences both for the economy and for human safety. Near the beginning of the book Rice praises Humphrey as an outstanding leader in terms of trying to get software engineers to use high quality and secure practices for building software.

The course then moves on to eXtreme Programming (for example, see the book by Beck [5]). This is an intentional and radical shift from the heavy duty data tracking of PSP to a highly iterative process that emphasizes collaboration and creativity. Indeed, Ken Schwaber (a key founder of the Scrum agile process) has called the conflict between the SEI-type developers (also called the waterfall process folks) and the agile developers "the War". Schwaber is not an advocate of war; he believes that peace is definitely possible between the waterfall process folks and the agile folks. Our discussion of "the War" helps students to see that there is no "silver bullet" in software engineering and that questions regarding what makes for a good software process are interesting and intellectually challenging. The team projects (discussed towards the end of this paper) are intended to force the students to take a stance on "the War" and attempt to produce a software process for an imaginary company that integrates ideas from the two sides in the conflict.

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, coding tests before the actual code is written, continuous integration, refactoring and the forty hour work week. XP exposes students to a recurring theme in the course: four eyeballs are better than two in order to prevent defects from being in the delivered code (an important argument for the use of pair programming). 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 agile processes (more generally) in industry. Important papers in this regard include the papers by Grenning [6] (relating to XP) and Schatz and Abdelshafi (relating to Scrum) [7].

The course then moves on to discuss the Capability Maturity Model (CMM) from SEI. We introduce 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, it introduces the 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. He argues that an organization that uses XP (and has the prerequisite management infrastructure in place) could certainly achieve CMM level 2 and could satisfy some of the key process areas for level 3 (and even one at level 5, the highest maturity level). Since Paulk published his paper, the CMM has evolved into the CMMI (Capability Maturity Model Integrated). We devote some class time to reviewing the revised CMMI process areas (the new terminology, as opposed to "key process areas") and their descriptions.

This leads into a more in-depth study of CMM 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 for giving developers a sense of ownership for the process, so that the developers do not view the process as something dictated from above. Our study of CMM (and CMMI) ends with a discussion of the latest data from the SEI which shows how the CMMI is 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].

It is interesting to note that almost none of the resources that we use to introduce students to software processes devote much attention to security concerns right up front. Indeed, even the CMMI does not focus on security per se (although a modified version of CMMI, called SSE-CMM, certainly does). The author tries to alert students to these issues early on, in an iterative fashion. The next section of this paper will discuss how security issues are integrated into the course.

The author used to include Open Source software development in this course, but due to time constraints, we no longer discuss Open Source in great detail. However, Rice [4] and other authors have serious concerns about the use of Open Source software in our critical information infrastructure.

Once the students have been introduced to several important software processes, they can begin to work with their teammates on the first team project. This project requires that they develop a documented software process for an imaginary company. (We shall devote more attention to this project in Section VI of this paper.)

III. BUILDING SECURE SOFTWARE

The course tries to alert students to the security issues in software development from day one. After our introduction to software processes (PSP, CMMI, XP, Scrum) we move on to discuss how security concerns can be integrated into the software development process. This portion of the course uses a high-level approach to this problem that emphasizes how security concerns should be integrated into the software development process during each phase of the process. We discuss how security can be incorporated into the requirements gathering, design, implementation, and testing phases of the project. We look at the basic security goals that apply to software development, including defense in depth, using community resources, and the principle of least privilege. We examine how these security goals might be in conflict with basic software project goals (like user friendliness and timeliness to market). These ideas are shared with the students, in part, in the author's lecture notes and Power Point slides which are made available to the students via Blackboard.

The introductory security lectures were greatly influenced by the excellent book by Viega and McGraw [11]. 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. We introduce other concepts from Viega and McGraw, including their stance against code obfuscation and their support for the use of good coding practices and good security testing tools. Once we have completed the overview of how security issues can be integrated into the software lifecycle, we turn our attention to a study that describes a project that very much follows the recommendations of Viega and McGraw and other experts relating to how security issues can be integrated into the software development process. This project is described in an interesting paper by Apvrille and Pourzandi [12]. Apvrille and Pourzandi describe how security issues were integrated into the various phases of the project that involved the development of an instant messaging service.

Our security lectures then focus on assessing software processes in terms of their effectiveness for producing secure software. We begin this discussion with the article by Noopur Davis et al. [13] that supports certain processes as being consistent with producing high quality and secure software. This article communicates the findings of the authors who worked teamed together as the Security Across the Software Development Lifecycle Task Force. This task force was organized by the Department of Homeland Security. Their purpose was to investigate this very important question: Which software processes are consistent with developing secure software? According to Davis and her co-authors, the good processes include TSP, Cleanroom Software Engineering and processes guided by SSE-CMM. XP and agile processes are not mentioned at all in this paper by Davis and her co-authors.

We then turn our attention to eXtreme Programming and the concerns that some authors have expressed regarding its appropriateness for developing secure software. In their book [11], Viega and McGraw devote just one paragraph to XP, voicing their opinion that it is not a good process for developing secure software. As mentioned in the previous paragraph, XP and agile processes were not mentioned at all in the paper published in IEEE Security and Privacy by the DHS task force.

This section of the course ends with an animated discussion about how the basic practices of XP could be modified to explicitly incorporate security concerns into the XP framework. Students are expected to incorporate what they have learned from this discussion into their second team projects (discussed in Section VI). In that project, students are expected to incorporate security practices into their imaginary company's documented software process. In some sense, the author views the in-class discussion of how XP can be made more secure "the climax" of the course. In many ways, the various topics we discussed were leading up to this question: How can agile processes like XP be improved in terms of producing secure software?

During our in-class discussion relating to how XP can be made more secure, the students always come up with some interesting concepts. These almost always include, the inclusion of a security engineer on the team, the use of security stories to help the security engineer to develop a security policy, incorporating security testing into the testing process, training all developers with regard to safe coding practices, and making sure that all security requirements are satisfied for a particular release before that release can be considered completed. Security-related quality assurance measures include code reviews by appropriate team members (i.e., those who have been

trained in secure coding practices) and the use of security-oriented static and dynamic testing tools.

An important issue brought up by Viega and McGraw in their book [11] is the role of threat modeling and risk assessment in a software process designed with security in mind. The author has incorporated a paper on this topic into the course (see Ingalsbe et al. [14]). Ingalsbe and his co-authors discuss how they are attempting to integrate threat modeling into software development processes at Ford Motor Corporation.

IV. ETHICAL AND PROFESSIONAL ISSUES

After our introduction to ideas relating to secure software development, we turn our attention to a discussion of professional and ethical issues in software engineering. The goal is to give 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 software project failures. Our discussion of the consequences of failure provides another opportunity for the class to appreciate the importance of producing secure software.

Our discussion of software failures begins with the Therac-25 accidents in which very subtle programming errors 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 [15]. We then discuss the Confirm fiasco, in which an integrated airline, hotel, and rental car reservation system was never completed because of the incompetence and one might argue the unethical behavior of project leaders [16].

The discussion of the Therac-25 and Confirm fiascos leads into a more general discussion of why software fails using the paper by Charette [17]. Although Charette does not focus on security issues per se, he does mention that the art of risk management is still in its infancy (according to many software professionals). According to Charette, many companies do not engage in any form of risk management. This again brings up the issue of threat modeling as discussed in the aforementioned paper by Ingalsbe et al. [14].

After Charette's paper on why software fails, we move on to discuss the Software Engineering Code of Ethics (Gotterbarn et al. [18]). Although the Code of Ethics does not focus on security per se, many of the issues in the code (including the use of good professional practices to assure the reliability and safety of software systems) certainly have a bearing on producing secure software. 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.

V. WORK CULTURE AND QA

The course then moves on to discuss a variety of work culture issues and quality assurance issues in software development. The discussion of work culture focuses on the idea of "congruence" presented by McLendon and Weinberg [19]. 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 (or, 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 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" [20]. 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.

The work culture issues have a bearing on the problem of producing secure software. If developers are not honest, if they are afflicted with personal demons, they are less likely to produce secure software. A congruent work culture, where developers can share their perspectives with honesty and without fear, are clearly important for producing secure software. This semester, we will end our discussion of workplace issues with a new and provocative paper by Robert Glass et al. [21]. Glass and his co-authors provide rather disturbing data about the pervasiveness of lying on software projects. Clearly, these work culture issues have important implications when it comes to developing secure software.

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 (e.g., see Ciolkowski et al. [22]. One interesting article that we discuss looks at the conflicts between software developers and software testers (Cohen et al. [23]). This illustrates the interaction between work culture and quality assurance issues in software development. The conflicts reported by Cohen and her co-authors resonate with the kinds of conflicts that have been reported between software developers and network security experts

in various organizations (for example, as described by Viega and McGraw in their book [11]). Another paper discussed in this portion of the course covers in some depth how testing was done in a large project that used an agile process (Talby et al. [24]). Finally, this semester, we intend to devote some attention to a new paper by Patrice Godefroid and his co-authors at Microsoft [25] concerning their research into static and dynamic software testing. These Microsoft researchers report that their new techniques have exposed many security vulnerabilities that had previously gone unnoticed.

VI. TEAM PROJECTS

The course involves two team projects. The first of these projects is the one mentioned briefly at the beginning of this paper. It is due about two-thirds of the way through the course. For this project, each team develops a defined software process for an imaginary company. Students are asked to integrate ideas from the various processes that we discussed in order to create a defined software process for their imaginary company. Students draw heavily upon agile processes and PSP in designing their company's documented process. The students are especially drawn to agile concepts, but they incorporate ideas (especially checklists and data collection forms) from PSP as well.

Last year (Spring 2008) one team did something that was truly interesting and somewhat unanticipated (from the author's perspective). They developed a documented software process for a company whose developers were located around the world. Just a few weeks after the author read this team's project, he attended the IEEE CSEET conference and he heard a fascinating plenary session talk by Bertrand Meyer on just this topic (see Meyer's Web site [26]). Consequently, the author plans to devote some time to the implications of globalization for software development this semester in this software engineering course. That is clearly a topic of growing importance and a topic that has serious implications for software security, as reflected in the articles by Iacovou and Nakatsu [27] and Ramingwong and Sajeev [28]. The CSI/FBI Survey [29] indicates that organizations do not outsource security functionality, yet the problem of assuring that software developed using offshore outsourcing is an important one. Iacovou and Nakatsu discuss the severity of communications and management issues in offshore outsourcing. The communications problems relate specifically to misunderstandings because of culture differences in understanding English (for example). The research of Ramingwong and Sajeev indicate that people in an Asian culture are more likely to remain mum (silent) when problems arise on a software project due to a variety of cultural issues. So, there are security implications in offshore outsourcing.

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 with a special focus on issues that relate to building secure software. The emphasis for these in-class team presentations is on creativity. While some teams end up giving a PowerPoint type of 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.

VII. CONCLUSIONS

The author truly enjoys teaching this software engineering course. 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 for job interviews. A year ago, a recent graduate sent the author an e-mail explaining that his company had adopted the software process that he and his team-mates had developed in the course being described in this paper! That team project was outstanding, a team-oriented process which borrowed some ideas from meta-Scrum. 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.

VIII. REFERENCES CITED

The PSP book [2] is a required book for this course. In addition, all of the references cited below are in the "coursepack" for the course (a list of articles the students access through the ACM and IEEE Computer Society digital libraries) except for references 4, 5, 15, 20 and 24.

[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] Rice, David, Geekonomics: The Real Cost of Insecure Software, Addison-Wesley Professional, Boston, 2007, 384 pp.

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

[6] Grenning, James, "Launching Extreme Programming at a Process Intensive Company." IEEE Software, November/December 2001, pp. 27-33.

[7] Schatz, Bob and Abdelshafi, Ibrahim, "Primavera Gets Agile: A Successful Transition to Agile Development," IEEE Software, May/June 2005, pp. 36-41.

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

[9] Pitterman, Bill, "Telcordia Technologies: The Journey to High Maturity", IEEE Software, July/August 2000, pp. 89-96.

[10] Guerrero, Felipe and Eterovic, Yadran, "Adopting the SW-CMM in a Small IT Organization", IEEE Software, July/August 2004, pp. 29-35.

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

[12] Apvrille, Axele and Purzandi, Makan, "Secure Software Development by Example," IEEE Security and Privacy, July/August 2006, pp. 10-17.

[13] 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.

[14] Ingalsbe, Jeffrey A., Kunimatsu, Louis, Baeten, Tim, and Mead, Nancy R., "Threat Modeling: Diving into the Deep End," IEEE Software, January/February 2008, pp. 28-34.

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

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

[17] Charette, Robert N., "Why Software Fails," IEEE Spectrum, September 2005, pp. 42-49.

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

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

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

[21] Glass, Robert L., Rost, Johann, Matook, Matthias S., "Lying of Software Projects," IEEE Software, November/December 2008, pp. 90-95.

[22] Ciolkowski, Marcus, Laitenberger, Oliver, and Biffl, Stefan, "Software Reviews: The State of the Practice, IEEE Software, November/December 2003, pp. 46-51.

[23] 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.

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

[25] Godefroid, Patrice, de Halleux, Peli, Nori, Aditya V., Rajamani, Sriram K., Schulte, Wolfram, Tillman, Nikolai, and Levin, Michael Y., "Automating Software Testing Using Program Analysis," IEEE Software, September / October 2008, pp. 30-37.

[26] Bertrand Meyer's web site: se.ethz.ch/~meyer

[27] Iacovou, Charalambos L., and Nakatsu, Robbie, "A Risk Profile of Offshore-Outsourced Development Projects," CACM, June 2008, pp. 89-94.

[28] Ramingwong, Sahgasit, Sajeev, A. S. M., "Offshore Outsourcing: The Risk of Keeping Mum," CACM, August 2007, pp. 101-103.

[29] Available at the Computer Security Institute web site www.gocsi.com