Protection Poker: Structuring Software Security Risk Assessment and Knowledge Transfer

Laurie Williams, Michael Gegick, and Andrew Meneely

North Carolina State University Department of Computer Science
{lawilli3, mcgegick, apmeneel}@ncsu.edu

Abstract. Discovery of security vulnerabilities is on the rise. As a result, software development teams must place a higher priority on preventing the injection of vulnerabilities in software as it is developed. Because the focus on software security has increased only recently, software development teams often do not have expertise in techniques for identifying security risk, understanding the impact of a vulnerability, or knowing the best mitigation strategy. We propose the Protection Poker activity as a collaborative and informal form of misuse case development and threat modeling that plays off the diversity of knowledge and perspective of the participants. An excellent outcome of Protection Poker is that security knowledge passed around the team. Students in an advanced undergraduate software engineering course at North Carolina State University participated in a Protection Poker session conducted as a laboratory exercise. Students actively shared misuse cases, threat models, and their limited software security expertise as they discussed vulnerabilities in their course project. We observed students relating vulnerabilities to the business impacts of the system. Protection Poker lead to a more effective software security learning experience than in prior semesters. A pilot of the use of Protection Poker with an industrial partner will begin in October 2008.

Keywords: software security, Wideband Delphi, Protection Poker, Planning Poker,

1 Introduction

According to the National Vulnerability Database[1], the number of reported vulnerabilities has increased six-fold (from approximately 1000 to 6000 per year) from 2000 to 2007. As a result, software development teams must place a higher priority on preventing vulnerability injection at each phase of the software life cycle. Because the focus on software security has rapidly risen only in the last decade, software development teams often do not have expertise in techniques for identifying security risk, understanding the impact of a vulnerability, or knowing the best mitigation strategy [15]. We propose the Protection Poker activity as a collaborative

[1] The Common Vulnerabilities and Exposures (CVE) and Common Configuration Enumeration (CCE) Statistics Query Page (http://web.nvd.nist.gov/view/vuln/statistics) provides the number and percent of a given vulnerability type reported for each year.

and informal form of misuse case[2] development and threat modeling[3] that plays off the diversity of knowledge and perspective of the participants. An excellent outcome of Protection Poker is that security knowledge passed around the team. Protection Poker is based upon an interactive effort estimation practice, Planning Poker[4] [8], that is used by many agile software development teams.

More important than the resulting effort estimate of Planning Poker is the discussion that takes place during the practice. Planning Poker provides a structured means for team members to educate each other, uncover hidden assumptions, raise awareness of issues and complications, and expose the range of alternatives of achieving desired goals. Protection Poker, which focuses on security risk assessment rather than effort estimation, provides this same structure for team conversation about security. Protection Poker is conducted in an iteration[5] planning-type meeting to yield a list of the overall relative security risk for each potential requirement for the iteration. After performing Protection Poker, participants can use this relative risk to determine the type and intensity of design and the validation and verification (V&V) effort that needs to be included in the iteration for each requirement. As a result, the necessary effort to implement the requirement securely is factored into the effort estimate.

Students in an advanced undergraduate software engineering course at North Carolina State University participated in a Protection Poker session conducted as a laboratory exercise. Through Protection Poker, students structured their discussion of the security risk of the requirements of their course project, iTrust[6], a role-based healthcare application.

The rest of this paper is structured as follows. Section 2 provides background and related work. We present the Protection Poker practice in Section 3. In Section 4, we discuss our experiences with the use of Protection Poker with advanced undergraduate students. We summarize and discuss future industrial trials of Protection Poker in Section 5.

---

2 A misuse case is a use case from the point of view of an actor hostile to the system under design. The goal of the misuse case is not a system function but a threat posed by that hostile actor. [1]

3 Threat modeling is a method for uncovering design flaws in a software component before the component is built [12]. Threat modeling emphasizes risk management at the architectural/design-level where risks are assessed and mitigation steps are outlined [19].

4 The "rules" of Planning Poker have no resemblance to the rules of the card game Poker except that each participant hides their cards from the other participants until a designated time. The creator of the Planning Poker likely chose this name to have the catchy effect of alliteration. Co-located teams often do use cards to do their estimation, though the cards must be made for the Planning Poker game to contain only selected values.

5 An iteration is a term often used by agile software development teams to mean a relatively short period of time (typically two to four weeks) during which a development team produces working code for a predetermined set of requirements [7]. Protection Poker can be used by non-agile teams during release planning.

6 http://agile.csc.ncsu.edu/iTrust/

2 Background and Related Work

In this section, we provide background on risk assessment in software security, Wideband Delphi estimation, and Planning Poker.

2.1 Risk Assessment in Software Security

The basis of software reliability engineering [17] is the acknowledgement that software is delivered with faults, but it is most important to prevent, find, and remove the faults that will most likely be encountered by users in normal operation. Similarly in software security, we acknowledge that software will be delivered with vulnerabilities, but it is most important to prevent, find, and remove the high risk vulnerabilities which are most likely be exploited by an attacker. If vulnerabilities must remain when the software is deployed, let them remain in low risk areas uninteresting to an attacker and least valuable to businesses.

Effective software engineering security practices include building security into the software product when finding and fixing problems is cheaper and more feasible than if done later in the software process [5]. Limited resources (e.g. time, money, and expertise) preclude software engineers from identifying and fortifying all security risks. Prioritization of software security efforts is most effectively informed by risk assessment of potential security problems [11, 12, 14, 15]. In Section 3, we suggest this risk assessment be structured via the Protection Poker practice.

Through software security [12, 14, 15] development practices, security is built into software rather than added after completion or delivery of the software. Security efforts during development should focus on the areas of a software system that are most likely to be attacked. A vulnerability[7] remains latent until an attacker has exploited the vulnerability to attack a software system. As software reliability engineering [17] places value on areas of code most used by customers, software security efforts should focus on the most risky areas of code that are most attractive to attackers and most valued by businesses. Value-neutral vulnerability-finding techniques, such as conducting the same level of penetration testing on all areas of the code, may cause the development team to expend valuable and limited security resources on low risk areas of the code that may already be adequately fortified, may be uninteresting to an attacker, or contain hard-to-exploit vulnerabilities. Software engineers can prioritize the verification and fortification of components to produce more secure software by considering risk.

2.2 Wideband Delphi Estimation

Wideband Delphi is based upon the Delphi practice [9], developed at The Rand Corporation in the late 1940s for the purpose of making forecasts. With the Delphi practice, participants are asked to make their forecast individually and anonymously

[7] A vulnerability is an instance of a [fault] in the specification, development, or configuration of software such that its execution can violate an [implicit or explicit] security policy [13].

in a preliminary round. The first round results are collected, tabulated, and returned to each participant for a second round, during which the participants are again asked to make a new forecast regarding the same issue. This time each participant has knowledge of what the other participants forecasted in the first round but not any explanation by the participants of the rationale behind their forecast. The second round typically results in a narrowing of the range in forecasts by the group, pointing to some reasonable middle ground regarding the issue of concern. The original Delphi technique avoided group discussion [4].

Boehm created a variant of this technique called the Wideband Delphi technique [5]. Group discussion occurs between rounds in Wideband Delphi; participants explain why they have chosen their value. Wideband Delphi is a useful technique for coming to some conclusion regarding an issue when the only information available is based more on experience than hard empirical data [4].

2.3 Planning Poker

In recent years, many agile software development [6] teams have estimated the effort needed to complete the requirements chosen to be implemented in an iteration and/or release via a Wideband Delphi practice commonly called Planning Poker [8]. Planning Poker is "played" by the team as a part of the iteration planning meeting.

With Planning Poker, the customer or marketing representative explains each requirement to the extended development team. We use the term *extended development team* (often called the "whole team" [2] by agile software developers) to refer to all those involved in the development of a product, including product managers, project managers, software developers, testers, usability engineers, security engineers, and others. In turn, the team discusses the work involved in fully implementing and testing a requirement until they believe that they have enough information to estimate the effort. Each team member then privately and independently estimates the effort in units of "story[8] points" (discussed more fully below). The team members reveal their estimates simultaneously. Next, the team members with the lowest and highest estimate explain their estimates to the group. Discussion ensues until the group is ready to re-vote on their estimates. More estimation rounds take place until the team can come to a consensus on a quantity of story points for the requirement. Most often, only one or two Wideband Delphi rounds are necessary on a particular requirement before consensus is reached.

Planning Poker ensures that all team members participate in the estimation process and that everybody's opinion is heard, regardless of whether they are among the loudest or most influential people in the group [10]. *The diversity of participant opinions about the effort required to implement a requirement drives the Planning Poker discussion*. A dysfunctional Planning Poker session is one in which participants decide to go with or are implicitly or explicitly coerced into agreeing with the estimate given by a person or persons determined to be most important or most

[8] In an agile software development methodology, a "story" is analogous to a functional requirement.

respected. As such, a culture in which a diversity of opinions is valued is necessary for Planning Poker to be an effective effort estimation technique.

In Planning Poker, estimation is based upon the notion of story points [7]. Story points are unit-less measures of effort relative to previously-completed requirements. The unit-less story points do not directly correspond to traditional effort estimates such as person-hours or person-days. As a result, estimation is generally done more quickly because participants focus on relative size and not on thinking about how long the work will take. The latter might depend upon which engineer is assigned the task and what their work schedule might be. The team can focus on the estimation with discussions like the following:

- "<requirement> is similar to <other requirement> which was a 5, so we'll give this a 5"; or
- "<requirement> is likely to take twice as long as <other requirement>"; or
- "<requirement> will take the entire iteration, let's give it an 8"

Team members are constrained to estimating from a set of possible story point values on an exponential scale (most commonly 1, 2, 3, 5, 8, 13, 20, 40 and 100) [7] that are the relative amount of effort necessary for the correct implementation, including software development, usability engineering, testing, and document authoring/updating. There are two reasons behind the use of a limited set of possible values. First, humans are more accurate at estimating small things, hence there are more possible small values than large values [7]. Second, estimation can be done more quickly with a limited set of possible values. For example, why argue over whether the estimate should be 40 or 46 when our ability to estimate such large requirements is most likely inaccurate?

The values are often calibrated such that a very small task is given the value of 1 and a value of 8 indicates that the requirement will take the entire iteration. The values of 2, 3, and 5 are given relative to these endpoints. A requirement which is given an estimate of more than 8 is referred to as an *epic* [7] and can remain an epic for a future iteration. Once an epic is to be implemented in the next iteration, the epic must be broken down into small independent stories with estimates of 1, 2, 3, 5 or 8. A team computes its velocity [2, 7]; *velocity* is a historical number of how many story points the team is able to implement in an iteration. In an iteration planning meeting, the team determines which requirements to implement in the next iteration by choosing the higher priority requirements whose story points fit within the capacity determined by the velocity estimate.

There are three major benefits to using the Planning Poker practice:

1. **Effort Estimate**. The team obtains effort estimates via the expert opinion of all the members of the extended development team. The incorporation of all expert opinions leads to improved estimation accuracy [10, 16], particularly over time as the team becomes experienced with Planning Poker.
2. **Estimate Ownership.** The estimate is developed collaboratively by the extended development team. Therefore, the members will feel the estimate is realistic and will feel more accountability since they own the estimate.
3. **Communication.** The conversations that take place during the process are useful for sharing knowledge and for structuring conversation between

those on the extended team with a diversity of perspectives. When one or more team members have a low estimate and others have a high estimate, team members have a very different perception of what is involved in the implementation and verification and/or have a range of technical knowledge or experience. As such, Planning Poker provides a structured means for:

- obtaining a shared understanding;
- exposing hidden assumptions of the technical aspects of implementation and verification;
- discussing the implications throughout the system for implementing a requirement;
- surfacing and resolving ambiguities realized via divergent perspectives on the requirement; and
- exposing easy and hard alternatives for achieving desired goals.

The first author has participated in more than a dozen Planning Poker sessions conducted with industrial teams. A subjective identification of the value of Planning Poker session would be distributed as follows:

- 20% of the value is the effort estimate obtained;
- 10% of the value is that the team feels ownership of this estimate; and
- 70% of the value is the communication that takes place in the meeting through the structured Wideband Delphi process.

The motivation behind conducting Protection Poker sessions is, likewise, to structure team communication focused on software security.

3 Protection Poker

> *Interestingly, it may be in this dearth of "qualified" people trained in security that a critical opportunity can be found. Though few practitioners have academic security training, they most assuredly do have academic training in some field of study. That means that as a collective, the computer security field is filled with* ***diverse and interesting points of view****. This is exactly the sort of Petri dish of ideas that led to the Renaissance at the end of the Dark Ages...* ***Diversity of ideas is healthy****, and it lends a creativity and drive to the security field that we must take advantage of.* *– Gary McGraw [15]*

We propose a Wideband Delphi, Planning-Poker type practice called Protection Poker that leverages a diversity of ideas, experience, and knowledge related to software security. The dual purpose of a Protection Poker session is (1) to structure a collaborative, interactive, and informal practice for misuse case development and threat modeling; and (2) to spread software security knowledge throughout a team. The output of a Protection Poker session is a list of the overall relative security risk for each potential requirement for the iteration. Protection Poker sets the stage for participants to use this relative risk to determine the type and intensity of design and the V&V effort that needs to be included in the iteration for each requirement. As a result, the necessary effort to implement the requirement securely is properly

estimated and an understanding is gained of the steps necessary for this secure implementation. This list of relative risk can be used to guide the prioritization of security engineering resources towards the areas of the software with the highest risk of attack. This prioritization and increased knowledge should lead a team toward the development of more secure software.

In this section, we present instructions for conducting a Protection Poker session. We then discuss how risk assessment and knowledge transfer is achieved through the use of the practice.

3.1 Protection Poker Instructions

Protection Poker is "played" during an iteration planning-type meeting. As with Planning Poker, the extended development team (including the product manager, customer representative(s), requirements analyst(s), usability engineers and marketing representatives) is involved. Iteration planning can be augmented by the participation of a security expert if one is available, though the presence of a security expert is not an absolute necessity. Our initial trial of Protection Poker in an undergraduate class, discussed in Section 4, demonstrated the power of collaborative discussions of security issues by non-security experts. These discussions can also be aided through the use of a checklist of security issues to consider.

With Protection Poker, the customer or marketing representative explains each requirement to the extended development team. Informal discussions of misuse cases and threat models ensue. For example, the discussion might reveal that the role-based access to some functionality needs to be more restrictive.

For Protection Poker, we use unit-less measures to compute risk. Risk is traditionally [3, 19] computed as in Equation 1:

Risk = (probability of loss)(impact of loss) **(1)**

We propose a variation on this general definition and compute the security risk for a requirement as computed in Equation 2:

Security risk = (ease of attack)(the value of asset that could be exploited with a successful attack) **(2)**

The value of a particular asset does not change based upon the various requirements that use that asset. Therefore, a relative value for an asset can be established for the system based upon unit-less/relative values of *value points*. For example, the team may decide that the password table is 100 times more valuable to an attacker than the table containing statistics of baseball players and the password table would get a value of 100 and the baseball player table a value of 1. A variation of the adage "When everything is high priority, then nothing is high priority" is "When everything is very valuable, then nothing is very valuable." As a result, the team is best served by actually differentiating the value of the assets used by the software system.

In an iteration meeting, the team members provide estimates for:

1. ease of attack (higher value indicates easier to attack) in unit-less/relative values of *ease points*; and

2. the value of the asset (in value points) being accessed by/protected by the program in that requirement. The value of the asset is based upon historical values for the asset or, if a new asset must be created to implement the requirement, based upon the team's discussion/voting.

The estimation can then focus on discussions like:

- "<new requirement> increases the attack surface[9] as much as does <other requirement>"; or
- "Both of these new requirements read and write to a table with credit card information"; or
- "Only the admin can execute this functionality."

Similar to Planning Poker, the team can only choose from a set of values, such as from 1, 5, 8, 13, 20, 40 and 100 (as are used in Planning Poker) for ease points and value points. The objective of limiting the choice of values is to significantly increase the speed of development of the security risk profile such that it can be done as a part of the iteration planning meeting. Our method allows the differentiation of ease points and of value points with seven possible values. NIST advocates this differentiation of risk be limited to High, Medium, and Low [19]. Trials of Protection Poker with industry will indicate whether this additional differentiation provided by the product of ease and value aids in the ranking of security risks and whether these seven possible values are more effective for differentiating risk for prioritization purposes.

Simultaneously, the team members reveal their estimates first for ease points and then for value points for each requirement. A *Protection Poker discussion is driven by the diversity of participant opinions on the ease of attack and the value of the protected asset for a requirement*. McGraw [15] calls this time of discussion ambiguity analysis. *Ambiguity analysis* is the subprocess capturing the creative activity required to discover new risks when one or more participants share their divergent understandings of a system [15]. Disagreements and misunderstandings are often the harbingers of security risk [15]. Through Protection Poker, these disagreements and misunderstandings are surfaced and resolved before a requirement is designed or implemented. As with Planning Poker, the team members with the lowest and highest estimates explain their estimates to the group. Discussion ensues until the group is ready to re-vote on their estimates. More estimation rounds take place until the team can come to a consensus on a quantity of ease and value points for the requirement.

Values for ease points and value points would need to be calibrated at the start of the project such that an implementation of a requirement that would be very difficult to attack because it does not increase the attack surface, is given a value of 1 and the an implementation of a requirement that would be very easy to attack be given a 40 or a 100. All other requirements are estimated relative to these endpoints. The calibration can change through the course of multiple iterations.

---

9 The attack surface is the union of code, interfaces, services, and protocols available to all users, especially what is accessible by unauthenticated or remote users [12].

3.2 Risk Assessment

The risk profile for a requirement is computed by multiplying ease points by value points, as shown in Table 1. The risk profile can be used to prioritize software security efforts, such as a more formal development of misuse cases or threat models; security inspection; security re-design; the use of static analysis tool(s); and security testing. The relative risk for a requirement and the resulting actions necessary to implement the requirement are factored into the effort estimate for implementing the requirement. The necessary software security effort can be factored into the overall effort estimate such that the resources are allocated and realistic completion times are committed for the implementation of a robust, secure, fortified requirement.

Table 1: Prioritizing risk with Protection Poker

Requirement	Ease Points	Value Points	Security Risk
Requirement 1	1	1000	1000
Requirement 2	5	1	5
Requirement 3	5	1	5
Requirement 4	20	5	100
Requirement 5	13	13	169
Requirement 6	1	40	40
Requirement 7	40	20	800

3.3 Knowledge Transfer

As part of the structured Protection Poker conversation, the extended team discusses the ease points and value points for each requirement, in turn. The team can share business details of the proposed requirements; such as the assets which will be utilized (e.g. sensitive, personal information in a database), and about the technical risks that jeopardize the business details. For example, a requirement that requires that 15 user input fields which are used in dynamic SQL queries be filled in by a customer is inherently more risky than a requirement that involves only batch processing. Similarly, high-usage requirements impose a Denial of Service risk. The structured discussion of security issues that occurs as a part of Protection Poker should greatly improve the team members' knowledge and awareness of what is required to build security into the product.

4 Experience with Protection Poker

Protection Poker has had an initial trial with a class of approximately 50 advanced (third and fourth year) undergraduate students at North Carolina State University taking a software engineering class. Prior to coming to a structured laboratory session

on software security, the students had one 50-minute lecture that provided an overview of software security. The lecture focused on input validation vulnerabilities including cross-site scripting, SQL injection, and buffer overflows.

In the laboratory, the students were given a vulnerable version of the open source iTrust role-based healthcare application. The students were asked to find vulnerabilities in iTrust, assign ease and value points to each vulnerability, and then enumerate the risk values in descending order to show that the top risks should be addressed first. The students spent 40 minutes searching for and identifying security vulnerabilities in iTrust. After a brief laboratory-wide discussion of the vulnerabilities found, students were given fifteen minutes to assign ease and value points to each vulnerability and then calculate the overall risk for that vulnerability. Two doctoral students studying software security, including the students' regular teaching assistant and his colleague, conducted the laboratory exercise. The primary educational objectives of the exercise were to (a) expose the students to common vulnerabilities in a system they are already familiar with; and to (b) elicit a discussion between the students who are more familiar with security with those who are not. While the hands-on activity of finding vulnerabilities helps students learn about security, both objectives were significantly reinforced by having the Protection Poker activity. As expected, the groups who discussed the relative risk of each vulnerability had disagreed in many cases. The group discussions that started from Protection Poker encouraged students to discover many different ways of exploiting their systems, leading them back to their computers to try to find more problems.

One of the most interesting parts of the activity was how fast general lessons about security were surfaced in the discussions. Independent of each other, many groups came to form general lessons to avoid future vulnerabilities. Among the lessons discussed were:

1. *Security cannot be obtained through obscurity alone*. Students had many discussions about using obscure (but not impossible to guess) information on their system for sensitive features (e.g. auto-incrementing database keys).
2. *Never trust your input*. Before the Protection Poker laboratory, most students never considered that their input could intentionally be malformed for the purpose of launching a security attack.
3. *Know your system*. For the first time in the course, many students felt the need to truly understand their production environment so that they could gauge ease points more accurately.
4. *Know common exploits.* Students realized on their own that as technologies evolve, so do the vulnerabilities.
5. *Know how to test for vulnerabilities.* The students discussed several strategies for adapting their automated tests to reveal possible vulnerabilities.

In addition to the general lessons learned, students discussed methods of discovering and exploiting vulnerabilities. Their discussions had a resemblance to misuse cases and threat modeling even though they were not taught either of these practices.

We observed that the Protection Poker activity provided far more discussion and learning about software security than in previous semesters without Protection Poker. In terms of lessons learned, teaching assistants usually expect one or two of the five

previously-discussed lessons to be learned as opposed to the five that consistently came up. Furthermore, students were more likely to listen to each other than listen to the instructors lecturing, since they were all peers in a group. Today's students (from the Millennial Generation[10]) learn better through discovery than by being told, and they prefer learning through participation rather than by learning by being told what to do [18].

The most positive part of the lab was that many students reported the experience as an "eye opener". Throughout the rest of the semester, discussions with the teaching assistants regarding major security design flaws of the system took place. Although the activity was simple and natural to the students, it provided a great way to expose students to the nuanced concepts of software security.

5 Summary

In this paper, we have proposed the practice of Protection Poker for leveraging the diversity of experience and knowledge about software security in a team. By the use of the practice, teams can collaboratively perform a software risk assessment of the requirements for their product and can share their software security knowledge with their teammates. The result of the use of the practice should be a reduction of vulnerabilities in the product through an overall increase of software security knowledge in the team. Protection Poker was piloted with a team of undergraduate software engineering students at North Carolina State University. We observed the students discussing misuse cases and threat models and sharing their limited software security expertise as they discussed vulnerabilities in their course project.

A pilot of the use of Protection Poker with an industrial partner will begin in October 2008.

Acknowledgments. This work is supported in part by the National Science Foundation Grant No. 0716176. Any opinions expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Funding for this work has also been provided by the Center for Advanced Computing and Communications (CACC, http://www.cacc.ncsu.edu/) and the Secure Open Systems Initiative (SOSI, http://www.sosi.ncsu.edu/).

References

[1] I. Alexander, "On Abstraction in Scenarios," *Requirements Engineering*, vol. 6, no. 4, pp. 252-255, 2002.

[10] Students born after 1982.

[2] K. Beck, *Extreme Programming Explained: Embrace Change*, Second ed. Reading, MA: Addison-Wesley, 2005.
[3] B. Boehm, *Software Risk Management*. Washington, DC: IEEE Computer Society Press, 1989.
[4] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches — A survey," *Annals of Software Engineering*, vol. 10, no. 1-4, pp. 177-205, November 2000.
[5] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
[6] A. Cockburn, *Agile Software Development*. Reading, Massachusetts: Addison Wesley Longman, 2001.
[7] M. Cohn, *Agile Estimating and Planning*. Upper Saddle River, NJ: Prentice Hall, 2006.
[8] J. Grenning, "Planning Poker or How to avoid analysis paralysis while release planning," https://segueuserfiles.middlebury.edu/xp/PlanningPoker-v1.pdf, 2002, accessed on February 26, 2008.
[9] U. G. Gupta and R. E. Clarke, "Theory and Applications of the Delphi Technique: A bibliography (1975-1994)," *Technological Forecasting and Social Change*, vol. 53, no. pp. 185-211, 1996.
[10] N. C. Haugen, "An empirical study of using planning poker for user story estimation," in *Agile 2006*, Minneapolis, MN, 2006, p. 9 pages (electronic proceedings).
[11] M. Howard and D. LeBlanc, *Writing Secure Code*. Redmond, WA: Microsoft Press, 2003.
[12] M. Howard and S. Lipner, *The Security Development Lifecycle*. Redmond, WA: Microsoft Press, 2006.
[13] I. Krsul, "Software Vulnerability Analysis," in *Computer Science*. vol. PhD West Lafayette: Purdue University, 1998.
[14] G. McGraw, *Building Secure Software*. Boston, MA: Addison Wesley, 2002.
[15] G. McGraw, *Software Security: Building Security In*. Upper Saddle River, NJ: Addison-Wesley, 2006.
[16] K. Moløkken-Østvold and N. C. Haugen, "Combining Estimates with Planning Poker – An Empirical Study," in *Australian Software Engineering Conference (ASWEC'07)*, Melbourne, Australia, 2007, pp. 349-358.
[17] J. D. Musa, *Software Reliability Engineering: More Reliable Software Faster and Cheaper*, Second ed. Bloomington, Indiana: Authorhouse, 2004.
[18] D. Oblinger and J. Oblinger, "Educating the Net Generation," Boulder, CO: Educause, 2005.
[19] G. Stoneburner, A. Goguen, and A. Feringa, "NIST Special Publication 800-30: Risk Management Guide for Information Technology Systems," no., July 2002.