Extending XP Practices to Support Security Requirements Engineering

Gustav Boström
SICS/KTH, Sweden
gusbo@kth.se

Jaana Wäyrynen
Stockholm University/DSV, Sweden
jaana@dsv.su.se

Marine Bodén
Ericsson R&D, Sweden
marine.boden@ericsson.com

Konstantin Beznosov
Philippe Kruchten
University of British Columbia, Canada
{beznosov, pbk}@ece.ubc.ca

Abstract
This paper proposes a way of extending eXtreme Programming (XP) practices, in particular the original planning game and the coding guidelines, to aid the developers and the customer to engineer security requirements while maintaining the iterative and rapid feedback-driven nature of XP. More specifically, these steps result in two new security-specific flavours of XP User stories: *Abuser stories* (threat scenarios) and *Security-related User stories* (security functionalities). The introduced extensions also aid in formulating security-specific coding and design standards to be used in the project, as well as in understanding the need for supporting specific Security-related User stories by the system. The proposed extensions have been tested in a student project.

Categories and Subject Descriptors
D.2 [**Software Engineering**]; K6.3 [**Software Management**]; K.6.5 [**Security and protection**]

General Terms
Management, Security, Human Factors.

Keywords
Security Engineering, Requirements, Agile Software Development, eXtreme Programming, Development methodology,

1. Introduction
For the last 20 years, developers of software intensive systems have applied the technical-rational approach to project management by using a sequential (or waterfall) lifecycle, including rigorous up-front planning, up-front design, and a constant care to monitor and drive the project to conform to the plan. On one hand this approach has served certain software-development activities well. Having planning and design artefacts available early on for experts to examine has facilitated the task of certifying system conformity to external standards, and has often made the life of the acquirer of software-intensive system easier. A discipline that took advantage of this sequential lifecycle model is *security assurance*, and this applies equally to safety certification in avionics or medical instrumentation.

On another hand, the failure rate of software projects is alarming. As shown in the CHAOS report [26], the actual success rate of software projects is very low: less than 50% success, and much of it due to management practices. Software design is more akin to research than to construction or manufacturing, and many of the management paradigms adopted form those engineering fields were simply not adapted to the needs of the software domain.

The problems within security engineering mirror some of the inherent problems with traditional, waterfall based and document-driven software development [10]. Existing security engineering standards are based on a sequential, non-iterative lifecycle and assume stable development environments where project plans and security requirements are defined, fixed and documented upfront. The ISO 15408 Common Criteria (CC) is an example [9]. The predecessors of CC were designed mainly for military use on the basis of the waterfall approach, with extensive documentation requirements as a result. CC inherited some of these problems and has also been criticised for being both a time and resource consuming process. Hearn is among those who claim that there is a marginal commercial interest driving the CC market due to the cost issue [15]. Even indirect costs, e.g. coming from the time devoted to producing documentation must also be taken into account [2]. Some attempts have therefore been made to produce a more flexible CC process [27, 28]. However, problems still remain.

The low success rate in software projects has spawned the emergence of a new breed of approaches to managing projects, known collectively as agile methods. These methods proceed iteratively: they rely on gradual emergence of the design and the requirements, and emphasize direct person-to-person communication rather than the heavy written documentation of the waterfall approach. These methods exploit the "soft" nature of software engineering and exploit many feedback loops in the process. Rather than "plan-design-build," the new methods proceed by "speculate-collaborate-learn" [14].

SESS'06, May 20–21, 2006, Shanghai, China.

These methods are showing some industry successes and seem indeed more suitable for modern software development. They have been considered for application in both security engineering [1, 5] and safety engineering [3, 23]. They also run counter to the accepted practices in security engineering, more specifically to security requirements engineering and security assurance.

Our previous work focuses on security assurance and examines how its practices fit or don't fit in the context of agile methods [6]. We also identify that one of the popular agile methods—eXtreme Programming [32]—lacks provisions for engineering security requirements, among other things [31]. A naïve way to address these issues would be to add more steps and document artifacts to the XP process to address the identified shortcomings. This would, however, reduce the benefits of using XP. An alternative method would be to find a way of marrying XP with the conventional practices of security requirements engineering. In this paper, we propose an approach to strike a balance between documentation-centric, plan-driven conventional practices of security engineering, and iteration-centric, feedback-driven XP practices. Our approach is derived from the results of a previous analysis of how XP deals with the security activities and requirements stated in the Systems Security Engineering—Capability Maturity Model and the Common Criteria [9, 31]. The approach combines the use of XP with activities taken from security standards, to better ensure that the security requirements are dealt with in the development of secure software.

In this paper, we specify our approach in detail. It extends the XP planning game process of identifying and prioritizing business requirements to include additional steps that result in two new security-specific flavours of XP user stories: *abuser stories* and *security-related user stories*. The former roughly correspond to threat scenarios, and the latter to the countermeasures for reducing the risks that are due to those threats. In addition, the introduced activities aid in formulating application-specific coding and design standards to be used in the project. The proposed extensions also help mapping between abuser stories, security-related user stories, and the coding and design standards. The mapping is intended to facilitate for developers and customers to understand the need for supporting specific security-related user stories by the system. Throughout the paper, we use examples from a project where security engineering students employed our approach to engineer requirements for a secure negotiation system they are developing.

The rest of the paper is organized as follows: Section 2 gives an overview of the XP planning game. Section 3 describes the proposed extensions to the XP planning game. Section 4 discusses related work, and section 5 draws some conclusions and sketches future work.

2. Overview of the XP Planning Game

To understand the extension it is necessary to understand how requirements are specified in XP. Below, follows a short description of the XP planning game.

As indicated by the name, the planning game involves planning, namely two types of planning. Firstly, release planning is used to create a system release plan, which involves identifying the overall system requirements and defining the scope of the whole project. The release plan is decomposed into the second type of plan, i.e. the iteration plan. Iteration planning involves revisiting the release plan, adding new requirements, selecting relevant requirements and refining requirements. More specifically, based on the release plan and possible new or changed requirements, iteration planning aims to define a detailed plan for each individual iteration. Because each iteration is short and ends with an executable version of the system, allows for quicker feed-back and help in ensuring that the project stays on track. Therefore, iteration plans are produced during a planning session which takes place just before an iteration begins, where possible new or changed requirements can be accounted for [32]. Requirements are defined as units of system features or functionality. In XP, they are specified in so-called User stories [4], which are written on index cards. An example of a User story is: *"A customer detail is shown by selecting it from a list."* User stories are expressed in short phrases and should be measurable and testable. An essential part of a planning meeting is negotiation, where the development team estimates each story in terms of ideal programming time, i.e. is how long developers estimate it would take to implement a particular user story including the tests [32]. The customer then decides what user stories have the highest priority to be completed. Based on the estimates provided by the developers, the priorities provided by the customer and the project time and resources available, developers and customers finally discuss and negotiate what should be implemented by moving the story cards around on a table or a whiteboard to create a release or an iteration plan.

3. Extending the Planning Game

Some XP proponents argue that the planning game process is sufficient for specifying security requirements as well. They see security requirements as any other requirement. We believe this is wrong. Therefore we propose to integrate security engineering activities with the planning game.

Having established a need for extending XP with security requirements engineering activities, it is necessary to state what the goals of these activities should be. An extension would be a lot less useful if it transformed XP into a heavy, document-centric, plan-driven process. The usefulness of an extension would however also be reduced if it did not incorporate essential security engineering activities. This balance of priorities is a delicate matter and the correct extension will depend on the situation of an actual project. Nevertheless it is useful to show how one such extension might look like. Based on previous research [31] and our experience with security engineering we believe these goals to be important:

- New activities should be aligned with XP practices and terminology as much as possible
- The activities should be amenable to iterative work, that is the outputs should be easy to rework and preferably not rely too much on documents
- The output of the requirements process should be easy to follow up during the ensuing testing and coding activities of the iteration
- Especially important parts of the output should also be adapted for external review
- Requirements gathering activities should encourage proactive definition of security requirements [21]
- Requirements should be based on risk analysis [30].

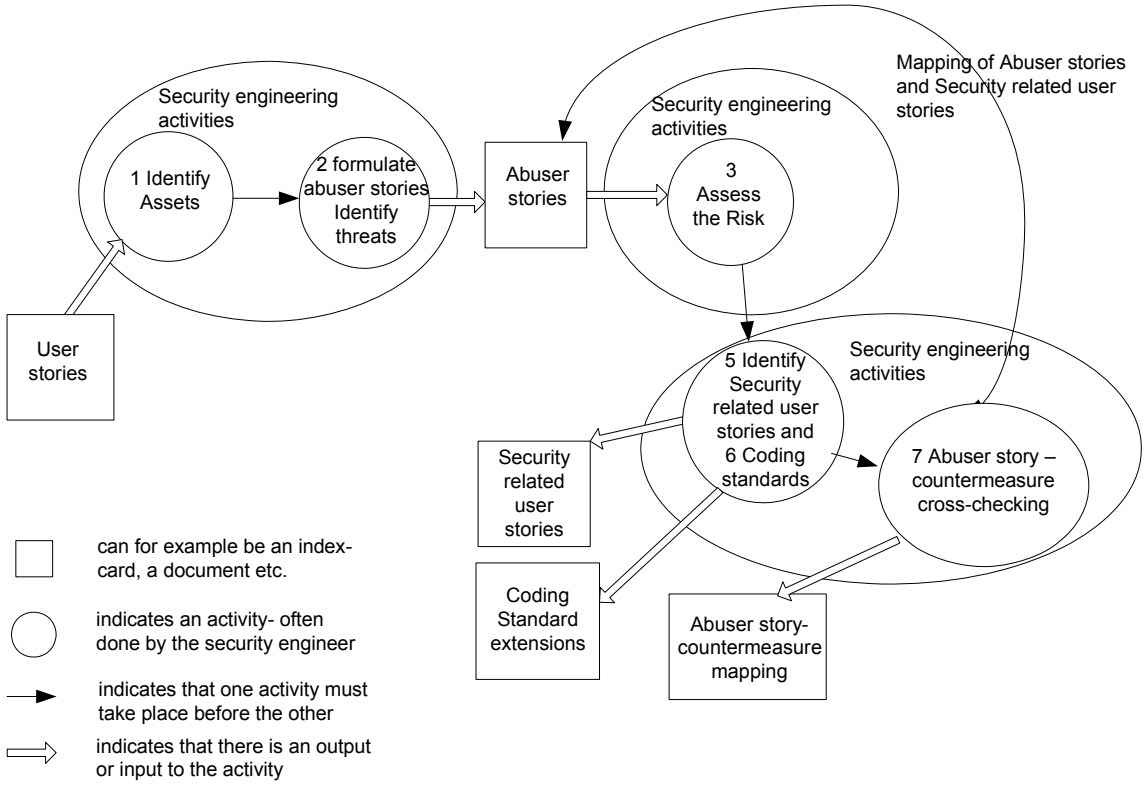

Figure 1. Overview of the extended XP Planning Game and its outputs

With these goals in mind, below is an outline of the steps we propose to be included in the XP Planning Game in order to engineer security requirements:

1. Identification of security sensitive Assets
2. Formulation of Abuser stories (Threat scenarios)
3. Abuser story Risk assessment
4. Abuser story and User story negotiation
5. Definition of Security-related User stories
6. Definition of security-related Coding standards
7. Abuser story – countermeasure cross-checking

These steps are illustrated in **Figure 1** and explained below. The activities are also illustrated with examples from artefacts developed when applying the process on a secure negotiation system.

1 - Identification of critical assets

Carried out concurrently with definition of the standard XP User stories, this step has the goal of gaining an understanding of what critical assets exist in the system under development. An asset is anything of value for the organisation or the users of the system. An example of an asset in the negotiation system could be: "*Confidential negotiation proposals*". The XP team, led by its security engineer, collaborates with the customer to identify relevant assets and their value. These assets are documented on the planning whiteboard.

2 - Formulation of Abuser Stories

In parallel with the identification of system assets, the security engineer (and possibly other development and customer team members) also develops scenarios for those threats that could result in the increased risks to the assets. This is a step where a security engineer is crucial because up-to-date fluent knowledge of security vulnerabilities, threats, and risks is essential for the identification of threat scenarios [12]. We call these threat scenarios *Abuser stories*, which we propose to use for representing Abuse cases [20] in XP projects. Abuser stories describe likely threats to critical assets in the form and language familiar to XP developers and customers. Like User stories, they are documented on index cards in a language understandable by the customer and developers. An Abuser story is a textual description of such a malicious interaction of a threat agent with the system that, if successful, results in the increase of risk to the asset(s) valued by the owner or user(s) of the system. An example of a simplified Abuser story is: *"A participant could modify another competitor participant's proposal to make it look bad"*. Abuser stories are discussed with the customer team to ensure their relevance and importance. Finding good Abuser stories is a brainstorming activity. However, using resources such as attack patterns can be helpful here [16]. Abuser stories are the basis for security testing of the system.

We argue that, unlike conventional threat analysis where threats are considered in broad categories and requirements are derived from them, specific, rather, than generic, Abuser stories should be used for engineering security requirements for XP projects. Our

argument is based on two points. First the mitigation of generic categories of threats cannot be tested. How can one test that no credit card information will be exposed to unauthorized users? If the developers cannot show the mitigation through test results, then they cannot objectively convince the customer that a particular category of threats will be mitigated. The second point of our argument is due to the distinction between the nature of problem and solution, respectively. Even though Abuser stories (the problem) are specific, the design of the system based on those stories (the solution) can be generic enough to mitigate classes of threats represented by the stories. It should be also noted that specific Abuser stories provide common measurable ground for the developers to receive the requirements in the Planning Game and to implement continues integration, and for the customer to gain confidence in the system, not just documentation.

3 - Abuser Story Risk assessment

When all Abuser stories and User stories have been identified the Risk assessment phase begins. In this phase the security engineer together with the customer team assesses the risk of the threats identified in the Abuser stories. The domain expertise of the customer team is vital here in order to estimate the business impact of threats [30]. Risk can roughly be seen as the product (threat probability x consequence) [30]. For each Abuser story, estimate the probability and the consequence of the threat being realised. The security engineer has the main responsibility for this, but it is the customers, who have the best knowledge of the environment and context in which the system will be used, so their input is also important. The risk assessment is documented by placing the Abuser stories on a quadrant according to the chances that they will occur and the consequences if they occur. For example, Abuser stories placed in the right top corner of the quadrant are associated with the highest risk for the customer's assets. When the assessment is finished, stickers with yellow or red colors are attached to the Abuser stories index cards to indicate their risk level. This activity is inspired by the experience of risk assessment activities at Ericsson and by the ISO 13335-2 standard for risk analysis [18].

4 - Abuser Story and User story negotiation

When risk assessment is finished the planning for the iteration begins. In this step the customer with the help of the security engineer has to decide on which Abuser stories to be countered in the iteration. This is analogous to how User stories are selected for implementation in a standard XP Planning Game. Abuser stories with high risk and consequences should be considered first. Here the developers together with the security engineer also have to estimate how much time will be required to counter the given Abuser story.

5 - Definition of Security-related User stories

Security-related User stories are functional security requirements for implementing countermeasures for the threats identified by the Abuser stories. Unlike Abuser stories however, security-related User stories can be validated directly by unit, system, and integration testing as other User stories. The security engineer defines security related User stories in cooperation with the development team and the customer. An example of a security-related User story is: "*Encrypt all communications: All documents sent between participants in the negotiation should be encrypted*". To determine the security functions that are necessary for countering Abuser stories, good knowledge of security architectures and mechanisms is necessary. A good help here can be to use security patterns [7]. This is a technical step which most people in the customer team are probably not interested in. Therefore, this activity could be carried out only with the security engineer and the development team. By defining security requirements as standard User stories it becomes clear for developers what kind of security functionality that needs to be implemented. In essence, this activity also serves as a requirements phase for the security architecture of the system. As the security architecture is something that could be of interest to an external security reviewer the Security-related User stories will be marked so that these requirements can easily be identified among the other Users stories. Security-related User stories are otherwise similar to other User stories.

6 - Definition of Security-related Coding standards

Not every Abuser story can be properly countered with a Security-related User story. Some important security threats are dealt with by system-wide properties of the system. One such example is binary code injection via buffer-overflow. This attack needs to be countered with secure coding techniques throughout the code base. Other attacks can be countered with design rules. Consequently, the Abuser stories could also result in an extension of the Coding standards that should exist in an XP project. An example of an extension to the coding standards could be: "*At no place in the code should any of the listed dangerous C-functions be used*". Once a new design or coding rule has been defined, it must be applied to the existing code base, and applied from this point on to any new code addition. Clearly, this is best achieved with the support of static code analysis tools and code reviews.

7 - Abuser story – Countermeasure cross-checking

To justify Security-related User stories, each story needs to be mapped into one or more Abuser stories. However, it is possible that an Abuser story does not have any corresponding Security-related User story. A Security-related User story is to be implemented in an iteration provided that the corresponding Abuser story has to be mitigated by the revision of the system developed in that iteration. This activity can be done by the security engineer but if any inconsistencies in the mapping are found, the customer (team) must be consulted to solve the problem. This step is slightly redundant because Security-related User stories are derived from Abuser stories from the start. However, it is so vital that the mapping of security functions and countermeasures to threats is correct that we believe this step is motivated. The cross-checking activity should also result in a specification of how Abuser stories, which are not easily dealt with through Security-related User stories are prevented. This is where the coding standards and the assurance activities for following them up are defined. This could include a specification of how activities, such as static analysis are used to verify that no buffer-overflows will occur in the system. Another activity could be code reviews through pair-programming with a security engineer. The output of this activity is a document outlining how each Abuser story is mapped to a countermeasure in the form either of a Security-related User story or verification activities.

3.1 Results from applying the process on student thesis project

As a first attempt to validate our approach we have applied the process on a student master thesis project. The task of the students is to construct a secure negotiation coordinator to be used for negotiating Service Level Agreements (SLA) electronically over the internet in virtual organizations. They used a whiteboard to document and discuss the different artefacts of the process. On the left side they put the user stories, the assets and the Abuser stories. On the right side, they place the results of the risk assessment. The different Abuser stories have been given numbers and these numbers are placed in the risk assessment matrix according to their risks and consequences. To the left at the bottom they place the Abuse story–Countermeasure crosschecking table. **Figure 2** shows some of the Abuser stories and Security related user stories produced.

When interviewing the students concerning their experiences of using the process a couple of interesting points were raised. The first was that they perceived the process to be significantly easier to work with than the Common Criteria (Primarily the students have mainly used CC in their course work.). However an issue that was raised was that it was difficult to perform the Risk analysis without a good description of the background and environment of the system. The students also chose to apply a more advanced method for the risk analysis activity. They applied the CORAS-method which they were familiar with [11]. This is an example of a possible extension of the process.

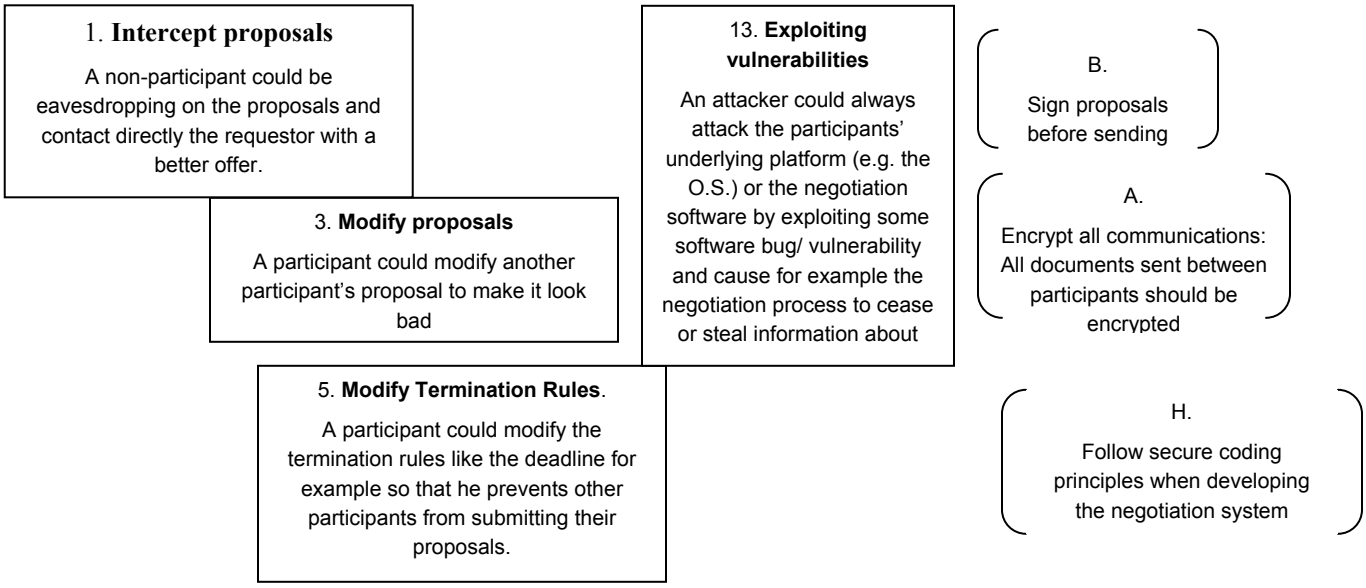

Figure 2: Sample Abuser stories and Security related User stories for a secure negotiation system

3.2 Implications for other Activities in the XP Process

Extending the XP Planning Game, as we described above, has impacts on other activities in the XP process. In the implementation phase of an XP iteration, the developers write tests and implement the selected User stories for the system (both standard User stories and security-related ones). The nature of these tests could be different when testing against Abuser stories. For example, it could be useful to complement unit tests and acceptance tests with automatic vulnerability testing, as well as static analysis of the system source code. Conventional assurance techniques, e.g., Red Team tests, where security experts explicitly try to perform attacks on the system, can be employed too. Coding activities are affected by the need to verify that the code complies with the defined coding standards. Static analysis tools could also be applied when doing Continuous Integration [13], for example.

When the system is handed off to the customer, the customer will have to have a clear understanding of the assumptions about the threats (and their agents) and the environment that have been made during the system development. The discrepancy, either pre-existing or emerged, between the assumptions and the reality would have to be handled by the customer by, for example, changing the environment or the system. Our approach does not propose methods for deriving such assumptions, which could be a subject of future work. We believe that this issue cannot be addressed by just requiring the customer to write the assumptions down, as the customer should not be expected to be sufficiently versed in security to document the assumptions about the expected threats and the environment. Even if the customer is capable of doing so, the assumptions in such form are not helpful for story-centric planning, testing, and implementation processes of XP. How would the customer verify and the developers make sure that the system corresponds to the assumptions? However, we envision that the process or the internal deliverables (i.e., Abuser stories, security design and Coding standards, Security-related User stories) of the modified Planning Game are sufficient for deriving, and if necessary documenting, these assumptions. This is where the expertise and skills of the security engineer in the XP team would become critical.

Similarly, the capability of the system to enforce specific types of security policies have to be well understood by the customer

before the system is deployed in production. It was beyond the scope of this work to suggest how the support for security policy types could be determined and/or communicated. We expect that the Abuser stories as well as Security-related User stories can be employed for this purpose.

3.3 Roles in the Security Requirements Engineering Planning Game

To clarify our proposed process adaptation, it is also useful to have an understanding of the roles different participants will have. All the contributors to an XP project are members of one project team that work together as much as possible [4]. In our definition, a project team consists of three interacting main parties, i.e., the customer team, the development team, and the security engineer(s). The customer team includes business representatives, e.g. domain experts, product managers and end-users. The customer team is responsible for providing the requirements and for defining the priorities of the project. The development team includes for example programmers, testers and system analysts. They provide the technical skills, but they also help the customer to refine the requirements and to define acceptance tests. The role of the security engineer can be played by one or several persons, who serve both the customer and the developers with security expertise throughout the project. More specifically, the security engineer primarily supports the customer during the requirements phase by specifying the security requirements. He or she plays the devil's advocate and identifies all the possible threats that need to be accounted for in the system's production environment. For developers, the security engineer provides support mainly during implementation where he participates in coaching and pair programming.

4. Related Work

Several attempts have been made to incorporate security activities into software development methods, e.g., by Breu et al. [8] and McDermott [19]. However, their work has a more formal character which could be more difficult to integrate with XP practices. The concept of Abuse cases however is very similar to what we propose with Abuser stories. Hope et al. also provide a good overview of Abuse Cases [17]. Beznosov & Kruchten identify points of conflict between agile development processes and conventional security assurance techniques [6]. Siponen et al. examine how security can be integrated into agile development methods in general, but do not target XP specifically [24].

A number of recent results are directly related to our work. Chivers et al. discuss how a security architecture can evolve iteratively [10]. Another interesting example of a development process with explicit support for security requirements is AEGIS, which builds on the spiral model for software development [12]. This process is similar to ours, but is not specifically aligned to XP or agile development. Vetterling et al. present a practical experience of applying the practices of the Common Criteria to software development projects [29]. However, we believe this approach to be an example of a plan-driven and more heavy-weight process and thus not suitable for agile development.

Peeters has a proposal for defining agile security requirements and he introduces Abuser stories [22]. He finds that Abuser stories have served him well in a number of projects. Our work is closely related to Peeters' but goes further by making it more explicit how Abuser stories fit into the overall development process. We add new steps to the process for risk analysis by defining Security-related User stories and cross-checking against Abuser stories.

The Common Criteria implicitly defines a process for requirement capturing which is applicable to a software development process. This is, however, not explicitly integrated into any existing software development method but rather treated as a separate activity. Moreover, the activities of threat identification and risk assessment are not covered in the Common Criteria because the standard relies on commonly used practices, defined in for example the ISO 13335-2 standard [18].

5. Conclusions and Future Plans

There is a tension between the needs of security engineering and the practices of agile processes. The former has long benefited from the waterfall lifecycle to "insert" security engineering, especially the requirements, early in the cycle, and to execute human and resource intensive activities only once in the cycle. The latter advocate few written work artifacts, and an iterative lifecycle that forces security-related activities to occur several times. However, security engineering can benefit from iterative development too [5], allowing a gradual discovery of security issues, and a progressive implementation of countermeasures. Iterative development will provide rapid feedback on the effectiveness of security requirement process and its implementation in the form of special user stories and design rules. By applying the simple documentation techniques of Agile methods and XP, such as index cards and whiteboard drawings, security work also becomes less heavy and the negative effects of needed rework in iterations is also reduced.

The dilemma between agile approaches and security concerns is not unique to security. Other quality attributes found in complex or mission critical systems are affected similarly: safety, high availability, or high performance. In these cases as well, requirements are not simply discovered by a dialog with the customer, and captured in User stories. Adaptation of the basic XP practices is required, a stronger emphasis on architecture and design standards is necessary, but these concerns can also be accommodated in an iterative lifecycle and a lightweight process.

Further work is necessary to validate our approach in practice, to determine how and how much security is affected by an agile approach. Another issue for further work is to elaborate on how security requirements are followed up using agile assurance techniques, such as pair programming and testing. A first round of validation is planned to be carried out through workshops in a Software Process Improvement Networks (SPIN-Stockholm) and through use of the process in more student projects. The results of the first student project showed that possible ways of improving our proposed process could be to include more support for reasoning about the system environment. It is also possible to enhance it by using more advanced risk analysis methods if necessary. But this should be seen as normal process adaptation.

In this paper we have shown how the elaboration of security requirements can be integrated with the XP planning game. It is our goal that by defining security requirements engineering activities in an agile manner, these activities will actually be carried out and not neglected altogether, as is so often the case today.

6. Acknowledgments

The example application of the process presented in this paper was partly funded by the European Commission through the IST program under Framework 6 grant 001945. The authors also wish to thank the master students for their valuable input derived from working with the process.

References

[1] Abrams, M. D., *Security Engineering in an Evolutionary Acquisition Environment*, in Proceedings of New Security Paradigms Workshop, Charlottsville, VA, 1998, pp. 11-20.

[2] Aizuddin, A., *The Common Criteria ISO/IEC 15408 The insight, Some Thoughts, Questions and Issues* Oct. 1, 2001. http://www.sans.org/rr/whitepapers/standards/545.php accessed June 17, 2005

[3] Amey P., and Chapman R., *Static Verification and Extreme Programming*. Proceedings of the ACM SIGAda Annual International Conference, 2003.

[4] Beck K., *Extreme Programming Explained: Embrace Change 2nd Edition.* Addison-Wesley, 2004.

[5] Beznosov, K., *eXtreme Security Engineering: On Employing XP Practices to Achieve "Good Enough Security" without Defining It*, in Proc. of First ACM Workshop on Business Driven Security Engineering (BizSec), Fairfax, VA, USA, Oct. 31, 2003.

[6] Beznosov, K. and Kruchten, P., *Towards Agile Security Assurance*, Proc. of the New Security Paradigm Workshop, White Point Beach, NS, 2004, ACM, pp. 47-54.

[7] Blakley B., Heath C. and members of The Open Group Security Forum, *Security Design Patterns*, The Open Group, 2004.

[8] Breu R., Burger K., Hafner M., Jürens J., Popp G., Wimmel G. and Lotz V., *Key Issues of a Formally Based Process Model for Security Engineering,* 16th International Conference on Software & System Engineering & Their Applications (ICSSEA), 2003.

[9] CC, ISO 15408 *Common Criteria for Information Technology Sec. Evaluation Version 2.1*, August 1999.

[10] Chivers, H. Paige, R., Ge, X., *Agile Security Using an Incremental Security Architecture*. Proceedings of Extreme Programming and Agile Processes in Software Engineering 6th International Conference, XP 2005, Sheffield, UK, 2005.

[11] CORAS, http://www2.nr.no/coras/, accessed in Jan. 2006

[12] Flechais I M., Sasse A. and Hailes S. M. V**.**, *Bringing Security Home: A Process for Developing Secure and Usable Systems,* ACM/SIGSAC New Security Paradigms Workshop, Switzerland, August 2003

[13] Fowler M. and Foemmel M., *Continuous Integration*. URL:http://www.martinfowler.com/articles/continuousIntegration.html. Accessed in January 2006.

[14] Highsmith J. A., *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, New York: Dorset House, 2000.

[15] Hearn, J., Does *the Common Criteria paradigm have a future?*, IEEE Security and Privacy, Vol. 2, Issue 1, 2004

[16] Höglund, G., McGraw, G., *Exploiting Software : How to Break Code***,** Addison-Wesley Professional, 2004

[17] Hope P., McGraw G., Anton A.., *Misuse and Abuse Cases*, IEEE Security and Privacy, 2004

[18] ISO/IEC, 4th WD 13335-2- *Information Technology- Security Techniques- Management of information and communications technology security- Part 2: Techniques for information and communications technology security risk management*.

[19] McDermott J. and Fox C., *Using abuse case models for security requirements*. Proceedings of the 15ths Annual Computer Security Applications Conferences (ACSAC). IEEE Computer Society Press, 1999.

[20] McDermott J. *Abuse-case-based assurance arguments. Using abuse case models for security requirements*, Proceedings of the 17ths Annual Computer Security Applications Conferences, 2003.

[21] McGraw G. and Viega J., Building Secure Software: How to Avoid Security Problems the Right Way, Addison-Wesley, 2002.

[22] Peeters J. *Agile Security Requirements Engineering.* Presented at the Symposium on Requirements Engineering for Information Security, 2005.

[23] Poppendieck M. and Morsicato R, *Using XP for Safety-Critical Software*, Cutter IT Journal, 15 (9), 2002, 12-16.

[24] Siponen M., Baskerville, R., Kuivalainen, T., *Integrating Security into Agile Development Methods*, Proc. of the 38th Hawaii International Conference on System Science, 2005

[25] SSE-CMM, *Systems Security Engineering Capability Maturity Model, Model Description Document Version 3.0*. URL: www.sse-cmm.org/model/ssecmmv2final.pdf. Accessed in January 2004.

[26] Standish Group, *The Chaos Report: Extreme Chaos*, West Yarmouth, MA: The Standish Group, 2001.

[27] *ST-Lite V 1.1*, July 2002 http://www.commoncriteriaportal.org/public/expert/index.php?menu=6 accessed 2006-02-01

[28] Fast Track, *Fast Track Assessment Methodology, Information Assurance and Certification Services (IACS), CESG*. http://www.cesg.gov.uk/site/iacs/index.cfm?menuSelected=3&displayPage=31 Accessed August 25 2005

[29] Vetterling M. and Wimmel G., *Secure Systems Development Based on the PalME project*, presented at Tenth ACM SIGSOFT Symposium on Foundations of Software Engineering, Charleston, South Carolina, USA, 2002.

[30] Verdon D, McGraw, G., *Risk Analysis in Software Design*. IEEE Security and Privacy, 2(4), 2004, pp.79-84.

[31] Wäyrynen J., Bodén M. and Boström G., *Security Engineering and eXtreme Programming: an Impossible marriage?*, XP/Agile Universe 2004, C. Zannier, H. Erdogmus, and L. Lindstrom, Eds. LNSC3134, Berlin: Springer-Verlag, 2004, pp. 117-128.

[32] XP, *Extreme Programming: A Gentle Introduction*. URL: http://www.extremeprogramming.org, Accessed in September 2005