Agile Development with Security Engineering Activities

Dejan Baca
Blekinge Institute of Technology
SE-371 79
Sweden
dejan.baca@bth.se

Bengt Carlsson
Blekinge Institute of Technology
SE-371 79
Sweden
bengt.carlsson@bth.se

ABSTRACT

Agile software development has been used by industry to create a more flexible and lean software development process, i.e making it possible to develop software at a faster rate and with more agility during development. There are however concerns that the higher development pace and lack of documentation are creating less secure software. We have therefore looked at three known Security Engineering processes, Microsoft SDL, Cigatel touchpoints and Common Criteria and identified what specific security activities they performed. We then compared these activities with an Agile development process that is used in industry. Developers, from a large telecommunication manufacturer, were interviewed to learn their impressions on using these security activities in an agile development process. We produced a security enhanced Agile development process that we present in this paper. This new Agile process use activities from already established security engineering processes that provide the benefit the developers wanted but did not hinder or obstruct the Agile process in a significant way.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management

General Terms

Security

Keywords

Agile Process, Software Engineering, Development process, Security Engineering

1. INTRODUCTION

For the last years, large parts of industry have shifted software development from a rigid waterfall to a more flexible Agile software development process. This shift is performed as and attempt to increase the effectiveness of software development. This is done by having a flexible structure with short development increments that handle change and new requirements that are easier than the older process. The Agile processes do impose limitations on the projects, it is no longer possible to create a complete picture of a product as all requirements are not yet known and no attempts are made to acquire this information. As stated in other literature studies [1] [2] [3], that compare security engineering (SE) [4] with Agile projects, this lack of a complete overview makes it harder and outright prevents some common SE practices from being performed in an Agile project. In this paper we interview developers that work in an Agile development process, with the basics of three mature SE processes [7]. With the interviews as a base we want to identify what parts of the SE processes are most compatible and beneficial to the project and what security activities the developers believe are not possible to perform in an Agile development process.

The aim of this study is to, through industry experience, identify what practices from mature SE processes are easily integrated and also provide a benefit to Agile projects. A suggested best practices based on the results will present an enhanced Agile development process that will have some of the security measures that SE processes relies on. We are also interested in why developers believe some of the practices do not work in an Agile process and what they believe prevents the integration.

2. RELATED WORK

Previous works in this topic have focused on litterature work and few studies have used industry experience and empirical data for strengthen their results. Hossein Keramati et al. [25] identify and evaluate security practices and activities in two SE process, Microsoft SDL [14] and Comprehensive Lightweight Application Security Process [24]. The paper presents an algorithm, called Agility Degree, for rating activities and its compatibility with an Agile process. Mikko Siponen et al. [5] identify the problems with integrating security in agile and devised an agile friendly method to identify, track and implement security features. Their method uses several known security engineering activities but is in the need of more practical experimentation to fine-tune the process. Beznosov & Kruchten [6] examined mismatches between security assurance techniques and agile development methods. The paper is based on literature studies and the authors identify some techniques that fit well with agile and others that are clear mismatches. Jaana Wäyrynen et al. [1] also conducted a theoretical analyses, comparing Common Criteria with Agile and determined they would benefit from

*ICSSP'*11, May 21–22, 2011, Waikiki, Honolulu, HI, USA

more imperial data to asses their results and premises. Laurie Williams et al. [8] devised a protection poker planning game that uses developer interactions and common Agile pratices to perform risk analyses during development. In the paper the author performs a case study at Red Hat IT and evaluates its risk analyses method.

3. BACKGROUND

In this section we briefly explain our generalization of the three examined SE process; Cigatel Touchpoints, Common Criteria and Microsoft SDL. The practices of these three processes were extracted by the author and explained as activities that should be performed in different phases of the development process. Based on the recommendation from the SE process each activity is also mapped to a specific phase of process. Four phases where identified as distinct steps, the boarders for these phases become vague in an Agile project but where well documented in the SE process. In order, the project is expected to pass these phases: requirements (Rq), design (D), Implementation (I), Test (T) and Release (R). The same information as in this section was presented to the interviews as explanations what a specific activity meant, what it provided and how it would be integrated into the process.

3.1 Cigatel Touchpoints

Cigital Software Security Touchpoints[9] has often been described as an lightweight SE process that integrates core activities in an existing development process and improves the quality and security aspect of the end product [10]. From the process we can extrapolate the following activities and claims within each phase:

- *Security Requirements (Rq)*: Security Requirements cover both overt functional security and emergent characteristics that are best captured by Abuse Cases and attack patterns.
- *Abuse Cases (Rq)*: Abuse Cases describe the system's behavior under attack; building Abuse Cases requires explicit coverage of what should be protected, from whom, and for how long.
- *Risk Analyses (D)*: Security analysts uncover and rank architectural flaws so that mitigation can begin. Disregarding risk analysis at the early phases leads to costly problems down the road.
- *Assumption Documentation (D)*: Designers, architects and analysts should clearly document assumptions and identify possible attacks.
- *Static Code Analyses (I)*: All software projects produce at least one artifact: source code. At the code level, the focus is on implementation bugs, especially those that static analysis tools, that scan source code for common vulnerabilities, can discover. Code review is a necessary practice, but not sufficient for achieving secure software.
- *Penetration Testing (T)*: Penetration Testing provides a good understanding of fielded software in its real environment. It does this by simulating real world working conditions and attack patterns.
- *Red Team Testing (T)*: Manual testing security functionality with standard functional testing techniques.
- *Risk Based Testing (T)*: Manual risk-based security testing based on attack patterns.
- *External Review (R)*: External analysis (outside the design team) that reviews existing touchpoints and performs their own.

3.2 Common Criteria

Common Criteria [11] is mature and well used SE principal that is ISO certified. A previous study has examine Common Criteria from an Agile perspective and identified activities similarly to ours [12].

- *Security Requirements (Rq)*: Identifying and documenting security and functionality for a given software project.
- *Agree on definitions (Rq)*: The first task for the organization is to define the stakeholders and to agree upon a common set of security definitions, along with the definition of the organizational security policies and the security vision of the IS. It is in this activity when the Vision Document artifact is created and should follow one of the available ISO standards.
- *Risk Analyses (D)* Risk must normally be determined from application to application. The final goal to achieve is the 100% risk acceptance. This is captured in the Risk Assessment Document, which is refined in subsequent iterations.
- *Critical Assets (D)*: This is where the Security Readiness Review (SRR) is used for the first time. It consists of the identification of the different kinds of valuable or critical assets as well as vulnerable assets by the requirements engineer.
- *UMLSec (D)*: Each asset is targeted by threat's that can prevent the security objective from being achieved. First of all, it is necessary to find all the threats that target these assets with the help of the SRR. In addition, it could be necessary to develop artifacts such as misuse cases or attack trees diagrams or UMLSec use cases and classes or sequence/state diagrams to develop new specific or generic threats or requirements.
- *Requirements Inspection (D)*: Requirements Inspection is carried out in order to validate all the generated artifacts and it is generated as a Validation Report. Its aim is to review the quality of the team's work and deliverables as well as assess the security requirements engineering process.
- *Repository Improvement (R)*: The new model elements found throughout the development of the previous activities and which are considered as likely to be used in forthcoming applications and with enough quality, according to the Validation Report, are introduced into the SRR. Furthermore, the model elements already in the repository could be modified in order to improve their quality. Thereby, all these new or modified model elements/artifacts, which have been introduced into the SRR, altogether constitute a baseline.

3.3 Microsoft Security Development Lifecycle Process

The Microsoft Security Development Lifecycle (SDL) [14] is a software development process used and proposed by Microsoft to reduce software maintenance costs and increase reliability of software concerning software security related bugs. It is based on the classical spiral model but there are attempts in making it more Agile friendly [15].

- *Security Requirements (Rq)*: Identify and enumerating security and privacy functionality for a given software project.
- *Role Matrix (Rq)*: Identifying all possible user roles and their access level to the software.
- *Design Requirements (Rq)*: Validate the technical design specifications and ensure they are appropriate relative to the Security Requirements for a given software project.
- *Quality Gates (D)*: Create appropriate security and privacy quality measures, including activities that need to be done for a fulfillment of requirement.
- *Cost Analysis (D)*: Analyses the cost implications of the different possible threats.
- *Threat Modeling (D)*: Create new threat models or validate existing threat models for correctness based on the products design. The threat model should identify security vulnerabilities.
- *Attack Surface Reduction (D)*: Reduce the design and end products attack surfaces, by limiting entry points and simplifying interfaces.
- *Security Tools (I)*: Use commercially available, open source and inhouse developed security tools to assist the project.
- *Coding Rules (I)*: Clarify the rationale for the deprecation of unsafe functions and help identify and recommend alternatives for these unsafe functions
- *Static Analysis (I)*: Have mandatory Static Code Analyses with predefined rules and priorities.
- *Dynamic Analysis (T)*: Use dynamic testing tools and perform an evaluation, triage the output, explain the results and develop a mitigation strategy for a given software program.
- *Fuzzy Testing (T)*: Use fuzzy test tools and perform an evaluation, triage the output, explain the results and develop a mitigation strategy for a given software program.
- *Code Review (T)*: Manual source code reviews of risk components.
- *Incident Response Planning (R)*: A response checklist that provides clear guidelines of action in the event of a security breach.
- *Final Security Review (R)*: A final, before release, security review that; reviews all threat models, validates security tool results, reviews all outstanding/deferred security bugs, reviews all exception requests as part of the security program.

3.4 Other

Besides the three SE processes there are some common knowledge security activity that are often recommended but are not distinct steps in any of the three SE process.

- *Countermeasure Graphs (D)*: An risk analyses method that focuses on identifying security features and prioritizing them [16].
- *Diff. review (I)*: Performing source code reviews on patches before the code change is committed to the source code repository [17].
- *Pair Programming (I)*: An Agile concept where developers code in a pairs. Solving and reviewing problems directly as the code is written [19].

4. RESEARCH METHOD

4.1 Case Study Context

As a complement to the process model description, the context of the study was as follows. Ericsson AB is a leading global company offering solutions in the area of telecommunication and multimedia. Such solutions include charging systems for mobile phones, multimedia solutions and network solutions. The company is ISO 9001:2000 certified. The market in which the company operates can be characterized as highly dynamic with high innovation in products and solutions. The development model is market-driven, meaning that the requirements are collected from a large base of potential end-customers without knowing exactly who the customer will be. Furthermore, the market demands highly customized solutions, specifically due to differences in services between countries.

4.2 Reseach Context

The process model used at the company is described and thereafter its principles are mapped to the incremental and iterative development; SCRUM, and Extreme Programming (XP). The model is primarily described to set the context for the case study, but the description also illustrates how a company has implemented an incremental and agile way of working. Due to the introduction of incremental and agile development at the company the following company specific practices have been introduced:

The first principle is to have small teams conducting short projects lasting for three months. The duration of the project determines the number of requirements selected for a requirement package. Each project includes all phases of development, from pre-study to testing. The result of one development project is an increment of the system and projects can be run in parallel.

The packaging of requirements for projects is driven by requirement priorities. Requirements with the highest priorities are selected and packaged to be implemented. Another criterion for the selection of requirements is that they fit well together and thus can be implemented in one coherent project.

If a project is integrated with the previous baseline of the system, a new baseline is created. This is referred to as the latest system version (LSV). Therefore, only one product exists at one point in time, helping to reduce the effort for product maintenance. The LSV can also be considered as a container where the increments developed by the projects

(including software and documentation) are put together. On the project level, the goal is to focus on the development of the requirements while the LSV sees the overall system where the results of the projects are integrated. When the LSV phase is completed, the system is ready to be shipped.

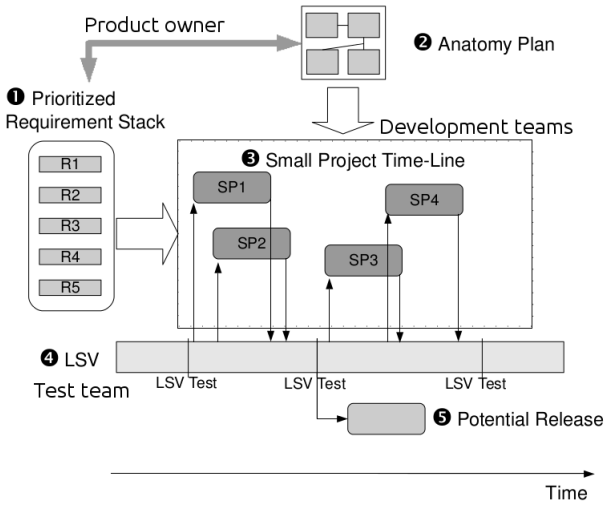

Figure 1: A quick overview of the Agile development process.

The anatomy plan determines the content of each LSV and the point in time when a LSV is supposed to be completed. It is based on the dependencies between parts of the system developed which are developed in small projects, thus influencing the time-line in which projects have to be executed.

If every release is pushed onto the market, there are too many releases used by customers that need to be supported. In order to avoid this, not every LSV has to be released, but it has to be of sufficient quality to be possible to release to customers. LSVs not released to the customers are referred to as potential releases. The release project in itself is responsible for making the product commercially available, performing any changes required to alter a development version into an release version.

In Figure 1 an overview of the development process is provided. The requirements packages are created from high priority requirements stored in the repository (1). These requirements packages are implemented in projects resulting in a new increment of the product. Such a project is refered to as a Small Project (3) and has a duration of approximately three months (time-boxed). When a project is finished developing the increment, the increment is integrated with the latest version of the system, referred to as last system version (LSV) (4). The LSV has a predefined cycle (for example, projects have to drop their components within a specific time frame to the LSV). From the LSV there can be different releases (5) of the system. These are either potential releases or customer releases.

4.3 Selection of Interviewees

The interviewees were selected so that the overall development life cycle is covered, from requirements to testing and product packaging. Furthermore, each role in the development process should be represented by at least two persons if possible. The following distinct roles were present:

- *Unit manager,* is responsible for distributing manpower and is responsible for any out of product projects, such as quality or process improvements.
- *Project owner,* has the overall responsibility of the product and has a final say in anything that affects his products projects.
- *Requirements engineers,* write the requirements and are part of the prioritization process.
- *Architects and experts,* solve design issues in the product and also participate in creating the anatomy plan.
- *Developers,* work in the small projects and implement and test their requirements.
- *Testers,* are part of the LSV team and focus only on verification and validation.

When selecting the interviewees we followed this procedure:

1. A complete list of people available for each subsystem was provided by management.
2. For each role of development at least two people were randomly selected for the study. An exception was made for unit manager and project owner as these two roles were occupied by one individual each.
3. The selected interviewees received an e-mail explaining why they have been selected for the study. Furthermore, the mail contained information of the purpose of the study and an invitation for the interview. Overall, 12 persons were contacted and participated in the interview.

4.4 Interview Design

The interview was divided into five parts, divided by the different phases of development; requirement, design, implementation, testing and release. The duration of the interviews was set to approximately one hour. Each part was first explained by the interviewer based on the information present in section 3. The interviewee could then ask more specific questions on subjects he was not familiar with. The interview was designed to collect issues and advantages from the interviewees. In order to collect as many issues as possible, the questions were asked from two perspectives; integrating an activity as part of core loop (cost) and the value of continuously using the activity (benenfit). The interviewees had to rate every activity in that phase, to rate an activity the interviewee had 10 points per activity in that phase. He could then distribute them freely between all activities. This voting scheme has the benefit of clearly weighting activity against each other instead of a more traditional rating each activity individuality. Also, the interviewees always had to state what their reason for their answer was and justify their ranking of the activities.

4.5 Threats to Validity

Threats to the validity of the outcome of the study are important to consider during the design of the study, allowing actions to be taken mitigating them. Threats to validity in case study research are reported in Yin [22]. The threats

to validity can be divided into four types: construct validity, internal validity, external validity and reliability.

Construct validity: One threat was the selection of people to obtain the appropriate sample for answering the research questions. Therefore, experienced people from the company selected a pool of interviewees as they know the persons and organization best. From this pool the random sample was taken. The selection by the representatives of the company was done having the following aspects in mind: process knowledge, roles, distribution across subsystem components, and having a sufficient number of people involved (although balancing against costs). Furthermore, it was a threat that the presence of the researcher influenced the outcome of the study. The threat was reduced as there has been a long cooperation between the company and university and the author collecting the data is also employed by the company and not viewed as being external. Construct validity was also threatened if interview questions are misunderstood or misinterpreted. To mitigate the threat an initial trial interview was conducted and the interview format was improved to be better understood.

Internal validity: The ratings collected by the subjects were construed with the internal validity in mind. By providing the subjects a fix number of points that they distribute between the different actives, we assure that the rating is actually comparing these specific activities and not other that some subjects might be aware of depending on their different experience levels.

External validity: The process studied was an adoption of practices from different general process models. Care was taken to draw conclusions and map results to these general models, to draw general conclusions and not solely discussing issues that are present due to the specific instantiation of the process at the studied setting. However, if one maps the general findings in this paper to other development processes their context must be taken into account. Furthermore, a potential threat was that the actual case study was conducted within one company. To minimize the influence of the study being conducted at one company, the objective was to map the findings from the company specific processes and issues to general processes. The characteristics of the context and practices used in the process are made explicit to ease the mapping.

Reliability: There is always a risk that the outcome of the study is affected by the interpretation of the researcher. To mitigate this threat, the study was designed so that data was collected from different sources, i.e. to conduct triangulation to ensure the correctness of the findings. The interviews were recorded and the correct interpretation of the data has been validated through workshops with representatives of the company. The study will also be continued with a practical study of the results. In the process of accepting the practical study the company examined and validated the results before coming to future studies.

5. RESULTS

First, project members grading of different security issues are reported and then this result is compared to the function of traditional software engineering processes.

5.1 Grading Security Issues

In all 12 professional project workers were asked to grade security issues with respect to requirement, design, implementation, testing and release within a project. First a within-subject analysis of variance, an F-test [18], was done for the different issues comparing costs and benefits. For costs significant differences were found within Requirement (0.001), Design (0.001), Implementation (0.01), Testing (0.05) and Release (0.001), i.e. there is a less than 0,1%/1%/5% chance that the differences are the result of pure chance. For benefit significant differences were found within Requirement (0.001), Design (0.05), Testing (0.1) and Release (0.05). These differences are further used for comparing the different issues within each step of the Agile development process.

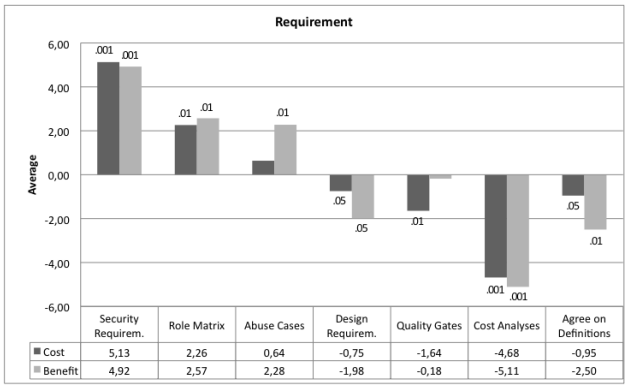

Figure 2: Above/below zero average for different requirements measured as cost and benefit.

The requirement phase is at the beginning of a project where the wishes of the stakeholder are gathered, analyzed, understood and specified. Figure 2 indicates that Security Requirement, Role Matrix and Abuse Cases are preferred against the other issues. To be more precise, the visual differences above/below average are confirmed by a Bonnferroni-Dunn [18] test showing statistical differences where above average show preferred issues (and below show disliked issues). The significant differences show that one (or more) issues are compared, not all contradicting issues, i.e. this is more of a guideline for further analysis.

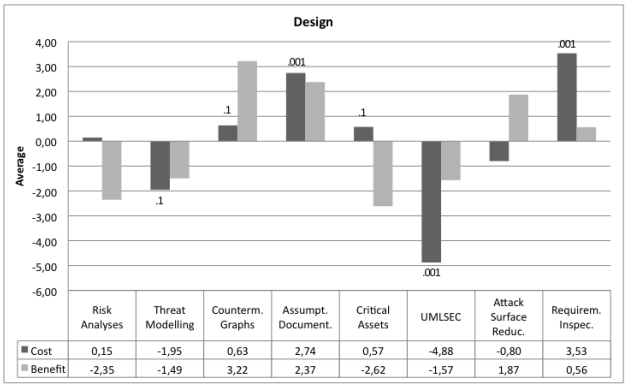

Figure 3: Above/below zero average for different design issues measured as cost and benefit.

The Design phase was explained, to the subjects, as the phase when the requirements are mapped into an architecture. A plan for the rest of the project is also finalized during this phase. Figure 3 indicates that Assumption Documenta-

tion Countermeasure Graphs, Requirement Inspection and Critical Assets are preferred against Threat Modeling and UMLSEC. Remark that all statistical differences originate from the cost comparison, i.e. it seems more difficult to compare benefits during the design phase. For the Critical Assets issue the results are contradictory with benefit just outside the (negative) 10% significant range.

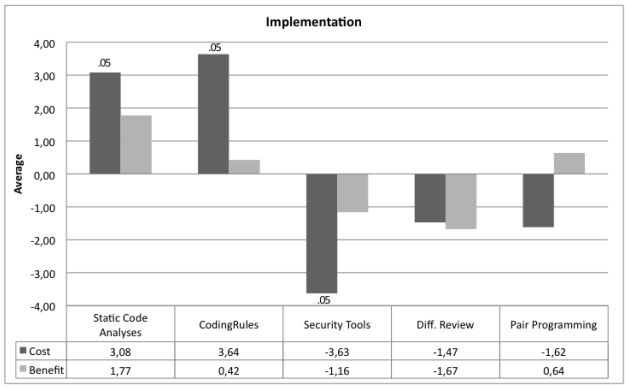

Figure 4: Above/below zero average for different implementations measured as cost and benefit.

During the implementation phase, see Figure 4, developers use the requirements and plan to write the source code and create the end product. Ericsson uses test driven development and as such does basic testing and test implementation during this phase. However, none of the examined SE processes have any specific test-driven activities during this phase. For cost Static Code Analyses and Coding Rules are preferred against Security Tools. While for benefits, no overall significant differences (F-test) were found which was also confirmed when the issues were individually examined.

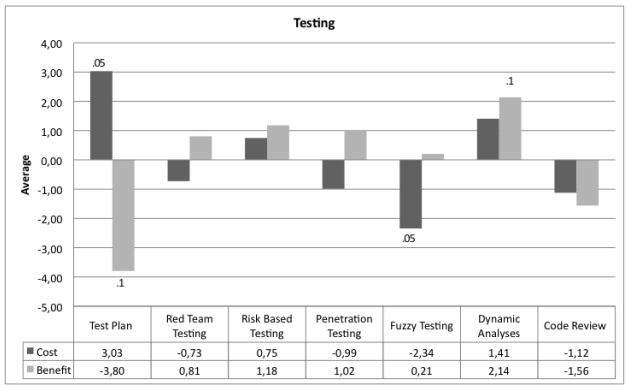

Figure 5: Above/below zero average for different testing issues measured as cost and benefit.

Testing includes all verification steps that are performed by the LSV test team. This does not include any basic testing the SP development teams perform as part of their small project time-line. The testing phase can be seen at the bottom of Figure 1. In Figure 5 Test Plan is preferred against Fuzzy Testing from a cost perspective, while Dynamic Analyses is preferred against Test Plan from the benefit point of view. So Test Plan is both preferred (cost) and disliked (benefit), i.e. it is cheap to introduce but gives little pay back. Obviously this is not a preferable issue for further consideration. Risk Based Testing is just outside the 10% scope for benefit making it a candidate for preferred issues.

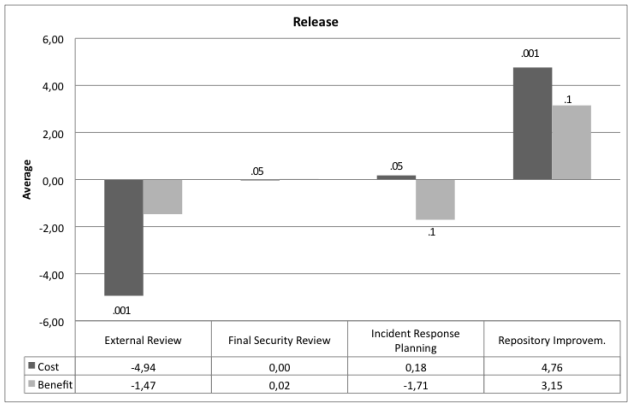

Figure 6: Above/below zero average for different release issues measured as cost and benefit.

In the release phase the product is completed for public access. Here, see Figure 6, Repository Improvement is better than all the other methods and contrary External Review is worse than all other for cost. For benefit Repository Improvement is better than Incident Response Planning.

5.2 Feedback Discussion

Using the interviews as a base we can create the framework of a new SE process that has the potential of being Agile friendly and create secure software. It is therefore important to examine what benefit the security provide and what is excluded from the process and why.

Writing Security Requirements was a strongly endorsed activity for integrating both cost and benefit. Developers believed that Security Requirements would identify what the easy gains are and guide the project right. Both Role Matrixes and Abuse Cases scored high and provided unique benefits to the process. Developers did however have a reservation with Role matrixes, some raised a concern that during the requirement phase it is still too early to have a grasp what roles will be in the end product. This concern applied more to new products then already mature projects. Developers also suggested that Abuse Cases should not be written during the requiriment phase but instead by the SP development teams. The remaining activities were deemed non-useful or to intrusive. Cost Analyses was the lowers ranked activity and developers believed it to not provide useful information and require decisions that are not yet made in an Agile project at the requirements phase. They also believed that nobody working with requirements would have the necessary knowledge to perform a cost analyses.

In the design phase we presented three different threat analyses methods, while the overall goal is the same. It should be noted that the Countermeasure Graphs were designed especially for Agile projects and in the same context as the interviewed developers. As such, the threat analyses value might be skewed. Threat modeling and UMLSec were both deemed too hard to implement. The last step in the Threat modeling activity, identifying vulnerabilities, was seen as to hard or impossible in this phase by most developers. Compared to countermeasure graphs were the last

analyses part is to identify how the product could or is directly stopping the attack. Because of the lack of a finished design, even after the design phase, developers believed that UMLSec would not be possible to perform or it would hinder the project too much. Attack surface reduction was seen as an effective concept but several developers raised concerns with the added complexity and cost of encapsulating and reducing the surface area at this stage of software development and instead advocated an ad-hoc solution where developers would raise the issue of attack surface reduction during stand-ups if it was required.

During implementation there were concerns that even Static Code Analyses do not provide any large benefit as few of the warnings reported by the Static Code Analyses tools actually produce any real bug reports. Previous studies [23] in the same company have shown that as low as 9% of the warnings can result into a bug report and even less are security related. Specialized security tools were seen as hard to integrate and also against the intentions of the Agile manifesto. Similarly Diff reviews where seen as hampering the projects flow or not providing any benefit. Diff reviews outside the team were seen as intrusive and cumbersome as the outside review is not familiar with the code. So, Diff reviews should be done within the team familiar with the code but developers now instead believed the code review to be ineffective as both reviewers have worked with the code, i.e. blind to defects in oneÂt's own code. An third alternative, the open source way, submitting patches to a component owner was dismissed because it would severly hamper the Agile flow. Pair programming, which is not a SE activity but instead belongs to SCRUM activities, had mixed responses.

Looking at the interview answers supplement the understanding of the main problem involved with testing. Red Team Testing was seen as too time intensive and would remove hours from the important LSV testing were the end functionality is validated and verified. Penetration Testing was hard to do because the project would not have a reliable environment to test the product in. This might be more specific to Telecom products because they require a more complex environment to operate compared to regular datacom products. Risk Based Testing made developers skeptic to its requirement of a risk analyses that most likely had been performed before the finished product had been created and which might have changed in design since. For fuzzy testing the developers were worried about the large number of test results and verification that would be needed after the automated fuzzing.

During the release phase it is clear that only Repository Improvements or lessons learned fitted well with the process. This is not surprising as the development process already implements this step for the general success of the development project. The other activities were not received well; especially the External security review was heavily criticized for interfering with the development process. The idea to not ship a release before an external party has reviewed the work is in harsh contrast to an Agile, quick communicating team.

5.3 Security Engineering and Agile Comparison

All tools or testing issues described above are collected from well known processes from Microsoft SDL (M), Cigatel touchpoints (T), Common Criteria (CC) and in a few cases from other processes used in the company environment (O). Those traditional software engineering processes mainly concern large waterfall projects instead of the today more common agile projects. In Table 1 significant differences in all test cases are described where + and - are positive and negative respectively. A + or - means any significance (0.001, 0.01 or 0.05) above or below zero average, i.e. the issue is significant different to at least one other issue within the investigated phase. Also, the comparison is based on a relative measurement separating one issue from other issues. If no significance is present among the involved issues, as in most benefit requirements, it is hard to draw any conclusions at all about the involved issues. Therefore, our conclusions mostly concern the cost aspects of the agile process. This is probably due to the interviewed subjects, because they work in a agile environment they are capable in estimating the cost of using an activity. However, because they are not security experts their knowledge in security is more diverse and therefore their answer divert so much that no answer is significant.

Requirement	*Cost*	*Benefit*
Security requirm. (T,M,CC)	+	+
Role Matrix (M)	+	+
Abuse Cases (T)		+
Design requirem. (M)	-	-
Quality gates (M)	-	
Cost analyses (M)	-	-
Agree on definitions (CC)	-	-
Design	*Cost*	*Benefit*
Assumpt. Document. (T)	+	
UMLSec (CC)	-	
Requirem. Inspec. (CC)	+	
Implementation	*Cost*	*Benefit*
Static Code Analyses (T, M)	+	
Coding Rules (M)	+	
Security tools (M)	-	
Testing	*Cost*	*Benefit*
Test plan (T)	+	
Fuzzy Testing (M)	-	
Release	*Cost*	*Benefit*
External Review (T)	-	
Incident resp. planning (M)	+	
Repository improvem. (CC)	+	

Table 1: Grading security issues within a project cycle. M - Microsoft SDL, T - Cigatel touchpoints, CC - Common Criteria and O - Others

6. DISCUSSION

In the discussion we will first compare how the developers believed the existing SE processes could integrate with Agile. This is followed by our own suggested processes that is based on the developers recommendations and our own judgment of required activities.

6.1 Evaluating Software Engineering Processes

Microsofts's SDL had the largest negative effect on an Agile development process. Developers where particularly negative to Cost analyses and the purpose of a Cost analyses is to focus the security push on the area that creates potentially the greatest cost and damage. However, developers did not believe that it would be possible to perform a sound Cost analyses at that early stage of development and that the benefit is no longer as good at the later phases of development. The lack of a Cost analyses is partly mitigated by having a more complex Risk Analyses and by the short iterations of a Agile process. Even if a development project has the wrong focus the sprint is short and the focus can be realigned after feedback from early customers or from retrospect meetings. Microsoft's Threat modeling (a form of Risk Analyses) was seen as too costly and a more solution based analyses was recommended instead. The concept of writing specific secure tools is a direct contradiction to the Agile manifesto [26] that states that collaboration and small teams should be prioritized instead of advanced tools. During testing the Dynamic analyses was seen as favorable. However, both fuzzy testing and a specific component code review where seen as costly too implement in the process. The code review is not as important for Agile teams because they collaborate more and have informal code reviews and feedback during daily stand-ups. The fuzzy testing does not have any equal improvement in the Agile process. Even though it is seen as a hamper to the process, projects will have to decide by them selves if Fuzzy Testing is an option. Lastly, Microsofts Incident response planning was not seen as a hinder but according to the developers it lacked the benefit for the work required.

Cigatel's Touchpoints did not have any negative activities during requirement, design or implementation phase. The basic Risk Analyses recommend by Cigatel can instead be replaced with a more Agile friendly alternative like Countermeaure Graphs [16]. However, during testing there are several test methods that raise the cost without providing a large benefit. The project in this study relies on test driven development and a large suit of automated tests for regression. The new automated test cases are also written at the same time as the implementation code. The Touchpoints testing methods all relies on having a ready product and then conduct a, mostly manual, test phase where the implementation code is not changed. This requirement by the tests methods makes them very Agile unfriendly, more specifically test driven unfriendly. An alternative would be to adapt or device a new test method that has the same benefit but uses test-driven and a large automated test suit instead of manual after release test methods. Lastly the concept of having an external, out of project, review is very hostile to a process that is depended on no bottlenecks or other blocking issues. An Agile retrospective meeting can in some sense replace a review but does not fulfill the requirement of being independent, e.g. external.

Common Criteria that is more of a metrics and measurements scheme does not have any implementation or testing activities. However, its activities for the products architecture assumes that there is an initial and final design created and then resulting in a implementation phase. For Agile projects the design phase if often very short and any design problems are solved during implementation by the team in a group effort. The group collaboration and daily stand-ups could replace the concept of a final architectural design.

6.2 An Agile Security Process

Product Owner	
Requirement	
Security Requirements	Role Matrix

SP Development Team	
Design	Implementation
Countermeasure graphs	Static Code Analyses
Assumption Documentation	Coding Rules
Abuse Cases	
Requirement Inspection	
Release	
Repository Improvement (retrospect meeting)	

LSV Test Team
Testing
Dynamic analyses

Table 2: Agile Security a basic Agile process with the most compatible SE activities.

If we combine the most compatible and beneficial activities, shown in Table 2, with an Agile process we can create an Agile security process that implements the most cost effective benefits from the three SE process. As we have discussed above some activities are not present at all but we have shown how an Agile process can cope without this activities but still have a high security standard. To better integrate the SE activities we have to move them from there recommended phase and placed them with the development team so the activity can be performed during a work package. Abuse Cases where moved to the design phase and the coding developers should write them instead of the requirement engineers. Also the release phase, Repository Improvement fits very well with the retrospect meeting that development team perform at the end of a work sprint. Integrating these two activities is made easier by moving them to a more developer heavy phase. An Agile process is a small team process that should focus its efforts on the Small Project teams and not have heavy pre- or post-coding phases. Comparing to Figure 1 most of the activities and security focus will be done by the development teams in step 3 of the process, the small project teams. During requirement the security enhanced Agile process primary focus is to write specific requirements for security goals, these will aid both programmers and tests to know what goals are required and need verification. A Role matrix aids the writing of user stories that programmers use to write the code. Having better security specified user stories makes it easier to write design safe software. The main focus for the improvements are with the developers teams and their work sprints. The design phase has several small planning and analyses steps that together create better security, while implementation focuses on automated tools that provide quick but rough code reviews. During the design phase the Agile process creates user stories and use cases to keep track of planned activities. Similarly Abuse Cases could be written for scenarios that may

not happen. Developers can then keep track on how they prevent the scenario or verify with basic test cases that it is not possible. The testing phase is performed by the LSV team and consists of other developers then the person that have written the code. Most of the test methods recommended by the SE process focus on manual testing. As such only the dynamic analyses scored high, since it does not require manual testing and uses the automated test suit.

7. CONCLUSION

Project developers from Ericsson AB, a global company offering telecom solutions, were asked to grade security improvement activities with respect to requirement, design, implementation, testing and release within a project using agile development methods. These security activities were chosen from three leading software development processes (Cigatel Touchpoints, Common Criteria and Microsoft SDL) in order to compare traditional large scale waterfall development security benefits with a smaller, more flexible, agile software development process.

All three traditional processes scaled badly to the chosen agile security processes because of too high costs or not being enough beneficial for the agile conditions, i.e. time constraints and reiterated increments of the product. Preliminary findings, based on the interviews, show that especially the design and testing phase scaled badly. Instead a new agile security process is proposed, integrating selected activities from the other traditional security development processes mainly into the development team but also to product owner and test team. In a future work these proposed security processes will be integrated and tested in a practical experiment with the existing development process.

8. ACKNOWLEDGMENTS

The authors want to thank Martin Hylerstedt for proofreading the text and Niklas Lavesson for valuable comments about the statistical analysis. We would also like to thank Ericsson AB for the opportunity to perform this study.

9. REFERENCES

[1] Wäyrynen, J.; Boden, M. Boström, G. Security Engineering and eXtreme Programming: an Impossible Marriage?, Extreme Programming and Agile Methods, Calgary, Canada, August 15-18, 2004

[2] Gustav Bostrom, Jaana Wayrynen, Marine Boden, Konstantin Beznosov and Phillippe Kruchten, Extending XP practices to support Security Requirements Engineering, ACM SESS 06, pp. 11-17, Shanghai, China, May 20-21, 2006

[3] Hossein Keramati, Seyed-Hassan Mirian-Hosseinabadi, Integrating software development security activities with agile methodologies, Computer Systems and Applications, ACS/IEEE International Conference on, pp. 749-754, 2008 IEEE/ACS International Conference on Computer Systems and Applications, 2008

[4] Davis, N., Secure Software Development Life Cycle Processes, CMU/SEI-2005-TN-024. Software Engineering Institute. Carnegie Mellon University, 2005.

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

[6] K. Beznosov, P. Kruchten, Towards agile security assurance, Proceedings of the 2004 Workshop on New Security Paradigms, September 2004.

[7] Bart De Win, Riccardo Scandariato, Koen Buyens, Johan Gregoire, and Wouter Joosen: On the Secure Software Development Process: CLASP, SDL and Touchpoints Compared, Preprint submitted to Elsevier, 16 January 2008.

[8] Laurie Willlliams, Andrew Meneely and Grant Shipley, Protection Poker: The New Software Security Game", Security & Privacy vol 8 no 3, 2010

[9] McGraw, G. Software Security: Building Security In, Addison-Wesley, 2006

[10] Steven, J., Adopting an enterprise software security framework. IEEE Security and Privacy, Vol. 4, No. 2, pp. 84- 87, 2006

[11] Feisal Keblawi, Dick Sullivan, Applying the Common Criteria in Systems Engineering, IEEE Security and Privacy, vol. 4, no. 2, pp. 50-55, Mar./Apr. 2006

[12] Mellado, D., Fernandez-Medina, E., Piattini, M., A comparison of the Common Criteria with proposals of information systems Security Requirements, The First International Conference Availability, Reliability and Security, 2006. ARES '06. 20-22 April 2006

[13] Beznosov, K., and P. Kruchten, Towards Agile Security Assurance, Proceedings of The New Security Paradigms Workshop ,White Point Beach Resort, Nova Scotia, Canada, 20-23 September 2004

[14] M. Howard and S. Lipner. The Security Development Lifecycle. Microsoft Press, 2006

[15] Bryan Sullivan, Streamline Security Practices For Agile Development, MSDN Magazine, November, 2008

[16] Baca, D. Petersen, K. Prioritizing Countermeasures through the Countermeasure Method for Software Security (CM-Sec), 11th International Conference on Product-Focused Software Process Improvement, PROFES 2010

[17] P. Rigby, D. German, and M.-A. Storey, Open source software peer review practices: A case study of the apache server, in Proc. of the International Conference on Software Engineering, 2008

[18] Ugarte, M. D., Militino, A. F., & Arnholt, A. T. Probability and Statistics with R. Boca Raton: Chapman & Hall/CRC, 2008

[19] L. Williams et al., Strengthening the case for pair programming, IEEE Software, vol. 17, No. 4, 19-25, 2000

[20] Schwaber, K., Beedle, M. Agile Software Development with Scrum. Prentice Hall, New Jersey, 2001

[21] K. Beck, Extreme Programming Explained: Embrace Change. Reading, Massachusetts: Addison-Wesley, 2000

[22] Yin RK. 1997. Case study evaluations: a decade of progress? See Rog Fournier, pp. 69-78 1997

[23] D. Baca, B. Carlsson, and L. Lundberg, Evaluating the cost reduction of static code analysis for software security, 3rd ACM SIGPLAN Workshop on Programming Languages and Analysis for Security 2008, PLAS'08, June 8, 2008 - June 8, 2008, Tucson, AZ, United states: Association for Computing Machinery, pp. 79-88, 2008

[24] Secure Software Inc., CLASP: Comprehensive Lightweight Application Security Process, http://www.securesoftware.com/process, Version 2.0, 2006

[25] Keramati, H.; Mirian-Hosseinabadi, S.-H.; , Integrating software development security activities with agile methodologies, Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on , vol., no., pp.749-754, March 31 2008-April 4, 2008

[26] Manifesto for Agile Software Development, http://agilemanifesto.org/