Introducing a Novel Security-Enhanced Agile Software Development Process

Martin Boldt*, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

Andreas Jacobsson, Malmö University, 206 05 Malmö, Sweden

Dejan Baca, Fidesmo AB, 111 39 Stockholm, Sweden

Bengt Carlsson, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

ABSTRACT

In this paper, a novel security-enhanced agile software development process, SEAP, is introduced. It has been designed, tested, and implemented at Ericsson AB, specifically in the development of a mobile money transfer system. Two important features of SEAP are 1) that it includes additional security competences, and 2) that it includes the continuous conduction of an integrated risk analysis for identifying potential threats. As a general finding of implementing SEAP in software development, the developers solve a large proportion of the risks in a timely, yet cost-efficient manner. The default agile software development process at Ericsson AB, i.e. where SEAP was not included, required significantly more employee hours spent for every risk identified compared to when integrating SEAP. The default development process left 50.0% of the risks unattended in the software version that was released, while the application of SEAP reduced that figure to 22.5%. Furthermore, SEAP increased the proportion of risks that were corrected from 12.5% to 67.9%, a more than five times increment.

Keywords: Software development, secure software development, secure agile development, agile method, software security, risk analysis, industrial setting, Ericsson AB

Introduction

Agile methods were initially seen as ideal for non-critical product developments. However, during recent years there have been several initiatives for the adaptation of agile methods for domains also in critical areas, such as, medical devices (Fitzgerald, Stol, O'Sullivan, & O'Brien, 2013)**Fel! Det går inrte att hitta någon referenskölla.** and money transfer systems (Baca, Boldt, Carlsson, & Jacobsson, 2015). That is, in situations where security risks are prominent characteristics. Despite these risks, most existing agile development methods hold an improvement-potential with regards to security aspects, e.g. the work by Cruzes *et al.* **Fel! Det går inrte att hitta någon referenskölla.** As a result, security is often added afterwards or included in the process by way of external resources. While there exist alternatives, such as, discrete security methods (such as checklists, management standards, etc.) that can supplement agile methods, few of these integrate into agile methods in a seamless, quality-enhanced yet cost-efficient manner.

In traditional software development processes, a security officer (also known as the product owner of security) performs audits or acts as a security advisor to the developers usually handles all matters related to security. However, in an agile development process, where iterations are short and changes made by the minute, it is not always possible to include a security officer from the outside, neither to integrate complementary discrete security methods in the process.

In terms of security, a general drawback with agile methods is the lack of a complete and updated picture of how all of the software requirements are implemented, which often results in difficulties to grasp let alone analyze the risks. Moreover, security flaws are often either overlooked or handled in an insufficient way. Due to the general work method in agile development, it is typically not possible to integrate systematic risk analysis in such a way that usable yet probable results are yielded. It is therefore important to predict how different ways of adding security aspects to agile processes will improve the security of the final product, in order to make sure that any additional costs due to the addition of security aspects are motivated by increased product security.

The work in this paper uses quantitative measures from risk analyses on a software product developed using an agile software development process. Thus, the risks identified in the risk analyses are used as indicators of the software product's security level, i.e. risk data was used as a way to estimate the end-product's level of security. In addition, the risk analysis data also reflects the efficiency of the agile team members in identifying risks in the software, as well as addressing them at an early stage. Although there exist other indicators of the level of security in software being developed (e.g., measures from security test cases or code reviews) that also could be used, those are considered out of scope for the focus of this study. Further, there is also a correlation between software security and software quality that must be taken into account. More precisely, there is a need for cost-efficient processes, tools, and guidelines for developing security-critical software in an agile way. In this work, our first task and contribution is to find a way to systematically integrate security management (including risk analysis) in agile software development project while at the same time meeting all the other software requirements with both quality and cost-efficiency. While our second task is to evaluate the security effects of the enhanced agile process in the development of a security-critical software product in an industrial setting.

Computer systems, network applications, and web services are vulnerable to attacks and need both protection of the involved assets and a way to deal with the consequences of bad, hazardous, or malicious practices. The work documented in this paper demonstrates how security features can be integrated into an agile software development method. The method has been tested at Ericsson AB, and in this setting and in accordance with the idea behind the method, a mobile-money transfer system that handles large amounts of monetary transactions has been developed. The mobile-money transfer system, which is a web-based service, intended to run as a stand-alone application on mobile units, is similar to online banking services in that it must be considered a highly attractive target for an abundance of malicious acts, such as, money laundering schemes, embezzlement attempts, and other criminal activities. In other words, security of the product is a highly prioritized concern.

This paper introduces a new security-enhanced agile software development process, hereinafter referred to as *SEAP*. A shorter version of this investigation along with a brief description of SEAP has previously been outlined by the authors

in (Baca, Boldt, Carlsson, & Jacobsson, 2015). Matters related to security are, in the case of SEAP, handled both as a process of integrating security in the agile development process and as a way of introducing specific security tools usable for the development teams. The introduction of integrated security tools facilitates a faster security analysis, a more detailed examination, and increased integration between risk analysis and security-enhancing measures, as well as, the overall software development. A main contribution in this work demonstrates how security can be integrated into an agile method in a real industrial setting, together with recommendations on how SEAP can be implemented in the development processes.

This paper is organized as follows. First, relevant literature is presented in order to further substantiate the novelty of SEAP. Then, the software development process is described and relevant constraints of our method and the setting of our studies are highlighted, i.e., at Ericsson AB. Thereafter, SEAP is outlined and critical tasks are introduced, such as, risk analysis, team configuration, and security management. After that, the risk analyses of the two agile methods (of which one is SEAP and the other is a default agile development approach deployed in the typical software development case at Ericsson AB) are compared. An analysis of the results of this comparison then follows. In the end, a discussion concerning the cost and utility aspects of the new improved agile method, i.e., SEAP, is conducted, and conclusions are drawn and future work is presented.

Related Work

Murphy *et al.* conducted a six-year survey at Microsoft, where they investigated the developers' attitudes with respect to agile adoption and techniques (Murphy, Bird, Zimmermann, Williams, & Nagappan, 2013). A main conclusion was that agile practices are problematic in areas relating to large-scale software development where the ability for agile practices to be used generally concerned all respondents, which may limit its future adoption. In a security-enhanced agile process, this aspect needs to be handled. Software developers need to ensure that their software product can withstand hostile attacks. Security-enhancing processes, including requirements analysis, threat modeling, static and dynamic code analysis, security audits, and product incident responses, can be embedded

into software development to make it more secure, cf. Microsoft's Security Development Lifecycle (Microsoft, 2017) or the Open Software Assurance Maturity Model (Chandra, 2017). Each security process has strengths and weaknesses, such as, the number and severity of security defects, how early in the development process they can be performed and the possibility to automate the process, etc. (McGraw, 2006). Nevertheless, software products require systematic security analysis and practices to encounter threats.

In the last decade, agile development has been preferred over more rigid methods, such as, the waterfall model. Petersen and Wohlin compared these models and indicated issues and advantages with agile software development in an industrial setting (Petersen & Wohlin, 2009). They identified a contradiction between using small sub-teams that increase control over the project and more complicated tasks on the management level where the coordination of the entire project takes place.

In Othmane *et al.* a method for security reassurance of software increments is proposed together with the integration of security engineering activities into an agile software development process (Bin-Othmane, Pelin, Harold, & Bhargava, 2014). The goal was to ensure the production of acceptably secure software increments that could be evaluated at the end of each of the iterations. This is similar to our study, but their work is carried out as a simple case study and without a real industrial setting.

The agile processes often impose limitations on the development projects (Siponen, Baskerville, & Kuivalainen, 2005). For instance, 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 the literature, e.g. Wäyrynen *et al.* **Fel! Det går inrte att hitta någon referenskölla.**; Keramati *et al.* **Fel! Det går inrte att hitta någon referenskölla.**; Davis **Fel! Det går inrte att hitta någon referenskölla.**, which compare security engineering **Fel! Det går inrte att hitta någon referenskölla.** with agile projects, this lack of a complete overview in agile approaches makes it harder and outright prevents some common security engineering practices from being performed.

As stated above, it is not obvious that more time and money also means a better and more secure code. For instance, Siponen *et al.* identify the problems with

integrating security in an agile setting and outline a method set to identify, track and implement security features by using several known security engineering activities (Siponen, Baskerville, & Kuivalainen, 2005). Although their method holds merit, they do not offer any practical experimentation to fine-tune the process.

Cruzes *et al.* compares how four agile teams in both Austria and Norway organize the security engineering process, with focus on security-testing (Cruzes, Felderer, Oyetoyan, Gander, & Pekaric, 2017). The results show a general lack of security-related competence and knowledge, which manifests by resting too much reliance on incidental pen-testing as well as unfortunate disregard of the possibilities from static testing for security. In summary, it is concluded that security-related aspects (in the four investigated agile teams) are not sufficiently handled and that security should be given more focus as well as effort in agile development.

Beznosov and Kruchten examined mismatches between security assurance techniques and agile development methods (Beznosov & Kruchten, 2004). Their paper is based on literature studies and the authors identify some techniques that fit well with agile development. Boström *et al.* also conducted theoretical analyses, comparing the Common Criteria to agile methods, and determined that they would benefit from more empirical data to assess their results and premises (Boström, Wäyrynen, Bodén, Beznosov, & Kruchten, 2006). For instance, Williams *et al.* devised a protection poker planning game that uses developer interactions and common practices to perform risk analyses during development (Williams, Meneely, & Shipley, 2010). In the paper, the authors perform a case study at Red Hat Inc., where agile approaches are used, and evaluate their risk analysis method. Baca and Carlsson proposed an agile security process in an industrial setting (Baca & Carlsson, 2011). Security issues were addressed using three well-known security tools; Microsoft SDL, Cigital Touchpoints, and the Common Criteria (Common Criteria, 2017). Test cases within imperative processes related to, e.g., requirement analysis, design, implementation, testing, and release, were investigated involving developers that were familiar with the agile project environment. One drawback with this investigation was that the security officer had an overall role in the project, i.e., it was more of an external control function than an integrated part of

the ongoing development work. Avoiding this drawback has been a primary goal of the development of SEAP.

Moreover, safety-critical computer-related failures are reported within a wide range of applications. For instance, Cotroneo and Natella describe an “unintended acceleration” issue in Toyota’s car fleet due to faults in the software (Cotroneo & Natella, 2013). Toyota had to recall almost half a million new cars and the mission-critical system was verified using techniques including static code analysis, model checking, and simulations. This example shows the importance of a systematic review of risks and their corresponding security measures in the development phase of new software, something, which has been taken into account in the design of SEAP.

Based on the existing related work a need for improving how security-related aspects are handled in agile software development processes has been identified. In order to address this identified research gap, SEAP is proposed as a way to distribute security-competence from the traditional use of a few security experts onto all developers instead.

The Industrial Software Development Process

Many applications that handle large financial transactions are tempting targets for various kinds of criminal activities, such as, fraud and thievery attempts. Some typical examples of systems that attract the attention of criminal actors include prepaid telecom systems, as well as, mobile and Internet banking services. These systems often have millions of users and penetrating their security mechanisms can provide direct economic benefit for any (malicious) user. The large financial values are not only tempting to the (malicious) end users; local banking and telecom agents with more extended access rights may also try to penetrate or circumvent the protection mechanisms in the system. Many systems, such as, those intended for mobile banking, use smart phones or tablets as access points. The software that runs in the terminals is literally in the hands of the attacker and this software is thus exposed also to reverse engineering attacks. This means that from a security perspective most of the critical access control rights should be implemented on the server-side and not in the terminals or tablets. However, moving most of the functionality from the terminal to the server could degrade

response time and increase network traffic. The software design should balance the need to prevent penetration attacks and attempts to circumvent the intended access control structure against the risk of low user perceived performance, quality of service, and excessive network traffic.

The industrial setting of the research documented in this paper is Ericsson AB, a leading global company offering solutions in the areas of Internet of Things, telecommunication and multimedia. Such solutions include charging systems for mobile phones, multimedia solutions, and network solutions. The market in which the company operates can be characterized as highly competitive and dynamic, and with a high speed of innovation-rate in both products and solutions. Typically, the software development model is market-driven, meaning that the requirements are collected from a large base of current end-users and potential customers, based on, e.g., statistics of their usage, error reporting, etc. Furthermore, the market demands highly customized solutions, specifically due to differences in services between countries. Thus, Ericsson AB often relies on agile development methods.

Extensive software projects that deploy a waterfall model have more and more been replaced by agile development methods fulfilling needs for faster and cheaper development. The process model used at Ericsson AB is described below and thereafter its principle practices are mapped to the incremental and iterative development of SCRUM and Extreme Programming, as they are also two important parts of the default software development process deployed at Ericsson AB.

Agile Practices at Ericsson AB

The overall direction of a project is set by the product owner while the detailed project scope of each implementation is planned, designed and set by the requirement board. This board consists of software managers that scope in requirements and try fulfill the request from the product owner. Due to the introduction of incremental and agile development methods at Ericsson AB, the following corporate specific practices have been developed. The first is to have small teams of 8-10 persons conducting short projects, i.e., lasting for a maximum of three weeks. The duration of the project determines the number of requirements selected for a requirement package. Each project includes all phases of

development, i.e., from pre-study via design and development to testing. The result of one development project is an increase of the system, which it is intended to be a part of. This means that several development projects run in parallel.

The packaging of requirements for projects is driven by requirement prioritization. Requirements with the highest priorities are selected and packaged for implementation. Another criterion for the selection of requirements is that they should 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 handled by the Quality Assurance Team (QAT). Here, only one product can exist at a specific point in time, thus helping to reduce the effort for product maintenance. The QAT is responsible for that 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 QAT focuses on the overall system, i.e., where the results of the projects are integrated and merged into a complete system. When the QAT has completed its phase, the system is ready to be shipped to the customers.

If every product release should be pushed onto the market, there would soon be too many products used by customers and thus in need of support. In order to avoid this, the idea is that not every single delivery from the QAT should be released, but even so, the aim is still that every product delivery should have undergone such a careful audit that it should be of sufficient quality to be possible to release to the market. In this case, it becomes a, so called, release candidate. A project that developed a specific product is in itself responsible for making it commercially available, and is thus also in charge of performing any changes necessary to transform a candidate version into a release version.

All security requirements included in this process are created and prioritized by the Security Officer, which is the final authority, i.e., the instance where the decisions on what should be developed in terms of security are made.

Figure 1: An overview of the default agile software development process used at Ericsson AB and as the baseline version of the study documented in this paper.

In Figure 1, an overview of the product release process is provided. The product owner, who has the overall responsibility of the product, in collaboration with an external customer (who represents the company that buys the product) is responsible for the business opportunity proposals (BOP), as shown in step 1 of Figure 1. A BOP describes new demands entering the system as a consequence of, e.g., new features. One BOP represents several business use cases (BUC) that are requirements of how to fulfill a specific BOP. Each BUC is then further translated into a technical and sequential description of how the BUC should be implemented, and in turn corresponds to one or several different user stories (US), which translate to a requirement containing information so that the developers can produce a reasonable estimate of the effort to implement it.

The requirements packages are created from high priority requirements by the requirement board. These requirement packages are implemented in subprojects resulting in a new increment of the product implemented by the development team, see (3) in Figure 1. Sub-projects within the development teams run for approximately three weeks. Each team handles design, implementation, and testing of the requirements. It might happen that a team starts in one sprint (a regular,

repeatable work cycle) to later be released in another. When a project team has finished an increment, it will be handled by the QAT as a release candidate. If additional features are dropped after a new sprint has been initiated, they will be tested in the next cycle, i.e., no new components/features can be added to a release candidate. Based on the work of the QAT, there can be different releases of the system. These are, as stated earlier, either release candidates or product releases on their way to the customers in the market.

In the default software development practices at Ericsson AB, all the security work is done by the security officer (also known as the product owner of security), as well as an external penetration tester once a year or once for each version of the product.

The Security-Enhanced Agile Software Development Process (SEAP)

As previously stated, there is at least one major practical drawback concerning the integration of security in agile software development processes typically deployed in industrial settings and it is the lack of a detailed overview of security issues. More exactly, how can flaws normally not visible in the development process be detected before a security incident. As an example, the security resource, often in the shape of a security officer, is not involved in the daily development within the different projects and often the project members do not have the necessary security skills to fix all detected bugs. The solution to this problem is to add security competence more closely to the actual software development in the agile process. So, in SEAP; as the agile structure is maintained, a consciousness of security is raised, as depicted in Figure 2.

Figure 2: An overview of SEAP. Security additions are shown in light green color to make comparisons between the two processes easier.

The requirement board includes a security architect who facilitates that tasks, initiated by the product owner, are properly handled as user stories (BUC). An initial task may be divided into several security-related user stories or redefined to fit a feasible security solution. The development teams implement one BUC by doing a risk analysis, implementing design rules (e.g. using static analysis tools) and performing a security code review. The security master is partially part of the development team acting as a security resource, i.e. bringing along security skills normally not found within an agile development team. This ensures a more functionally secure product because of both more skilled programmers as well as mutual executed code reviews. The pen-test validation should focus on finding future security failures, besides the more traditional functional tests.

A security group acts as the main resource for managing tasks related to risk and security in the software development process. This means that the group acts outside, e.g., the development teams, but at the same time serve as a readily

accessible resource. The group maintains different competences represented by the following four security roles.

Roles in SEAP

The first role, the *product owner* is the person within the company that is responsible for the products towards upper management. He or she is responsible for the direction of the product and how the finances should be distributed. Further, the product owner has the final say in most issues with the products, from development to sales, but usually does not have the detailed understanding as the people working with the specific topic.

The second role is the *security officer* who handles traditional features connected to security, such as, ISO-certification, legal aspects, and other non-technical issues connected to the product owner. The security officer is also responsible for the BOP including initiation, integration, and prioritization of requirements in the product development process.

The next role, the *security architect*, is responsible for transforming BOPs into more technical description of BUCs by looking at, e.g., user interface, entry points into the application, functionality of the actual application, configuration, and product deployment. These requirement and design issues are then sent to development teams that initiates further design, implementation, and testing activities. Normally, each project team is assigned one BUC and each developer one US, or if a US is split into several tasks; then several developers could work in parallel within the same US. The role of the security architect is also responsible for coordinating the security masters. Compared to a traditional software architect, who is responsible for making sure that the different parts of the project fit together, the security architect instead focuses solely on security aspects. It is thus the security architect's responsibility to ensure that the security of the product increases over time, primarily using risk analysis measurements, design rules, code reviews and penetration tests.

The next role, the *security master*, is responsible for the security activities performed during sub-project development within the BUCs. In practice, several development teams share a security master working full-time within the project. The security master participates in various security activities throughout the

product's development lifecycle, e.g., risk analysis, code reviews, maintaining coding rules and a verification system, and educate/coach developers in the teams. The development teams conduct risk analysis activities in collaboration with the security master, manage questions related to security guidelines, and handle endpoint security. More practically, the members of the development team write, test, and, verify security features. They also conduct structured negative test cases, explore functional tests, and create fuzzy test cases to be run by a quality assurance team. Additionally, it is the security master that performs follow-up code reviews on the implementation of high risk requirements.

The final role is the *penetration tester* that handles questions related to security guidelines as part of the test analysis. The development teams have a common security view, which is derived from the end-user's perspective. This means that the average team conducts exploratory testing and runs fuzzing tools. Note that earlier phases of penetration testing (in premature product versions) can be done from outside the project and then sometimes by using external consultants. In SEAP, the penetration testing is done by the QAT as part of the (internally) project. Normally, a penetration tester relies on risk analysis results, and then verifies that the system is secure by default. Automated tools for validating system security are of course often used in such a process. Also, black box testing is a frequently utilized method in order to find hidden or newly introduced vulnerabilities.

Apart from these roles, SEAP also includes the traditional roles of *software developers* and *testers*. In Table 1, a short description of the roles included in the SEAP method is outlined.

Table 1. Description of the different roles used in SEAP.

Role	Description
<i>Product owner</i>	Responsible for the product towards upper management, which includes the general direction of the product and related financial aspects.
<i>Security officer</i>	Overarching responsibility for the product's quality in terms of security. Decides on which standards/certifications to implement and how much resources that should be spent on increasing security.
<i>Security architect</i>	Responsibility for assuring that the overall security of the product increases over time. Writes the scope for penetration tests conducted by the QAT.

<i>Security master</i>	Acts as information security hubs that assist the developers in solving security-related issues. They are also responsible for carrying out the risk analyses and that risks are correctly addressed within the same sprint.
<i>Penetration tester</i>	Relies on an end-user perspective when performing penetration tests to find security weaknesses.
<i>Software architect</i>	Overarching responsibility for ensuring that the product is well-designed, e.g., that the various features work together in a coherent way.
<i>Developer</i>	Implements features in the product and carries out the risk analyses with support from the security master.
<i>Tester</i>	Verifies the implemented system features. Creates security test cases together with the security masters based on the result from risk analyses.

Activities in SEAP

The purpose with the risk analysis is to estimate the likelihood and degree of negative consequences caused by attacks to the product. The outcome is the determination of a risk value that, depending on its severity, calls for a certain degree of deployment of security-enhancing measures. In the implementation-phase, a partial code review is done under the governance of the security master. During this review, a static code analysis tool helps the developers to find and fix security flaws. In the verification phase, the results of the test run by benchmark tools are analyzed, penetration tests are performed, which finally makes sure that identified guidelines and best practices are followed. The security master is responsible for a security information-sharing hub, which all software developers have access to. Since the security master is physically located in the development teams, he/she can easily answer questions and provide support. In the default agile development process deployed at Ericsson AB, security issues are handled by the security officer as that person typically is the only security expert that the project team has access to. So, in SEAP, security issues are handled by the security architect (BUC and US), security masters (risk analysis, design rules and code review) and internal penetration tester (penetration test validation).

The security group, that consists of everyone with a dedicated security-related role in the project, has the possibility to stop a delivery from a development team if:

- a risk analysis or code review indicates severe problems, or
- there are known warnings, which are ignored, i.e., not corrected, or
- the implementation does not fulfill targeted best practices and security goals.

Also, new security requirements may demand a part release, new release, or to be an emergency release. The latter implicates that all ongoing development will be stopped until the emergency risk level is reduced.

The Risk Analysis Component

One of the biggest differences between the default agile development process and SEAP is the risk analysis component. Traditionally, a risk analysis was conducted at the start of each major release of the product. Within SEAP, risk analyses are instead performed per BUC, even though there may be several BUCs per release. This creates an incremental risk analysis approach that better fits agile development praxis.

A security master must assume that an attacker can outwit and bypass thoughtful protection and may anticipate vulnerabilities that have not yet occurred, i.e., those that are outside the scope of the current risk analysis. Risk can be defined as the combination of the probability that a threat will occur and the possible consequences it may impair, cf. Peltier (Peltier, 2010). In combination, this means that risk analyses should include not only known or established risks, but also newly discovered risks and risks that change over time, i.e., the probability and/or the consequences changes due to the protection included in the system or product under review. In the industrial setting of our study, the risk analyses are executed by the development team, typically headed by the security master. Each risk analysis is done during approximately one hour for a specific BUC within a sprint.

In our case, the members in the developments teams are well-suited for performing the risk analyses that are carried out throughout the development project. Security masters educate the developers in security topics in the daily implementation work in each sprint, i.e., as security-related aspects arise during the development. Allowing developers in a team to discuss and reflect on hands-on security problems together with the team's security master can result in continuous

education in the handling of security problems, a more crisp identification of suitable solutions, as well as, leading to important insights on best-practices for the developers. This on-going co-education among the developers is a significant feature of SEAP.

The inclusion of security masters in the development teams are needed because it is not practically possible to make every developer a security expert on his/her own. Therefore, security masters are necessary to (1) assist and educate the team members in security aspects, and (2) to accumulate security-related knowledge in order to spread this throughout the entire development process and to all involved parties. By considering both qualitative aspects and quantitative measures derived from the identified risks in the risk analyses performed in accordance with SEAP, it is possible to evaluate the increase of knowledge in security among the developers. The bottom line is that security awareness and knowledge should be implemented at the lowest level in the development teams, rather than at the highest level by the security officer or some external consultant, as was the case in the traditional agile development process at Ericsson.

Risk Estimations

In the risk analysis of SEAP, the probability of a successful attack has a range between 1 and 5, where 1 signifies a low likelihood of an attack, and 5 a high probability of an attack. Similarly, the impact of the attack also has a range between 1 and 5, where:

1. Minor damage to the system or service from internal (safe) source and data.
2. Minor damage to the system, service or data from external source, e.g. network connections.
3. Denial of service for other users or circumvention features.
4. Disclosure of confidential data, destruction of data, escalated privileges or financial fraud.
5. Full system (or component) compromised or undetected modification of data.

The impact ratings shown above were added by the developers of the mobile money transfer system. All impact ratings were created to be as concrete and

specific as possible so that any developer easily can rate risks with the use of those ratings.

Within this risk analysis approach, the estimated probability, between 1 and 5, is multiplied with the estimated impact, also expressed between 1 and 5, to determine the combined risk value, i.e., the higher the value, the greater the risk. For risk values below a certain threshold, the team does nothing unless the cost is negligible or a synergistic effect exists, i.e., the error can be handled in the context of a more serious correction. If the risk value is above the threshold, measures should be taken to correct it, resulting in an improvement in the product's security state. A risk may of course also be ignored if, e.g., the threat is not technically possible to address from the local sprint group's point of view or if the cost for mitigating the risk exceeds the utility of the security fix. The threshold levels are unique for each product and is decided by the security officer.

Actions Based on the Risk Analysis Results

A risk analysis is executed once per BUC during the design phase of the software development process. Each risk analysis focuses on the requirements of that specific BUC only. Depending on each obtained risk value and the risk's relevance to the BUC, one of three actions is determined:

- carry out the correction within the BUC as an US that mitigates the risk, or
- create a new BUC, delaying the correction for future versions, or
- accept the risk.

Risks that are corrected as US become the responsibility of the development team. The security master might aid the team with the task to manage it, but it is the team who owns the risk and thus is responsible for lowering it. So, a technical solution is proposed and implemented with the aim to mitigate the risk.

Accepted risks either have too low of a risk factor to be corrected and/or the team does not see any technical solution that can mitigate it. The security master is responsible for the risk and if the risk is severe enough, the security master might document mitigating factors that operators of the products should have knowledge about.

If a risk is severe, but the correction is out of scope of the BUC, then the security master will discuss the risk with the security architect that can create a new BUC to implement the mitigating features. The initial release might be susceptible to the risk, but a future version of the product will thus contain corrections that prevent the risk from being realized. Due to the short iteration of an agile development process and the early detection of the risk, it is easier to plan and implement the risk mitigation in the BUCs.

The Development of the Mobile Money Transfer System

In the part of development of the mobile money transfer system, where the traditional agile development process was deployed, 86 persons distributed among 8 software development teams, requirement boards, etc., were included (see Figure 3). Out of these 86 persons, one had the role of the security expert. Note that staff in the QAT were not included in this study as they did not work with relevant tasks, i.e., they had no effect on the risk analysis results. Both the baseline development process and the versions that deployed the SEAP method involve penetration tests, but neither one of those are part of our study as they do not affect the risk analysis results. In any case, the development teams of course had to follow Ericsson AB's general development requirements including, for instance, those that specify interaction with end-users, as well as, those that concern legislative demands.

The main difference between the project included in this study and an ordinary agile project at Ericsson AB is the amount of resources spent on explicit security competence. Usually, there is one security officer taking care of all security issues, e.g., making sure that the product meets security standards, as well as, initiating security tasks within the agile process. In this work, we include the first version (1.0) of the Mobile Money transfer product, which was developed according to Ericsson's traditional agile development methodology, i.e., the inclusion of one full time security officer, who is responsible for all security aspects.

Figure 3. A comparison of the number of security resources in (a) the baseline agile development process and (b) SEAP. The dotted red arrows indicate to whom the security-related questions are targeted, e.g., in SEAP most questions are handled by the security master in each development team.

In versions 2.0, 3.0, 4.0, and 5.0 of the same product, an extended agile development process with further emphasis on security was deployed, i.e., this is four iterations of SEAP. The security-enhanced process increases the security resources to 4 full-time employees per project (see Table 2). The main argument for making this change is the security-criticality of the software, i.e., as a high level security was necessary, more of the developers needed to be involved. Figure 3 shows the distribution of these extra resources.

This study focuses on the risk analyses headed by the security masters in the development teams, leaving the performance of the full development process out of scope for this investigation. However, to enable a full understanding of the process, the total amount of security resources should also be considered.

Table 2. The amount of security resources used within the software development process during the development of five separate versions of the mobile-money transfer system.

Version 1.0	Security officer	1 person á 100 %
Version 2.0-5.0	Security officer	1 person á 100 %
	Security architect	1 person á 100 %
	Security master	3 persons á 67 %, in total 200 %

In both the baseline development process and those based on the SEAP method, the security officer assists the product owner to understand aspects about security on the product owner level, e.g., with respect to legal and certification aspects, see Table 1.

In SEAP, the security architect helps the *requirement board*, which consists of software managers that scope in requirements, to understand aspects about security on the BUC-level, e.g., aspects concerning software security requirements or design. Further, a person holding the role as security master is assigned per two or three teams (25 % per team) to assist the developers with a general understanding of security and how to address such problems on a technical US level. Security masters are also responsible for carrying out code analysis, e.g., using static analysis tools and manual code reviews on security-exposed sections of the code. Available time apart from security work, usually about half of the time, should be spent as a regular software developer in a team. This is important since it allows the security master to continuously keep track of the security status of the on-going work. This also provides the teams with an easy access to a knowledgeable person regarding security aspects. In total, these additions of security competence in SEAP, compared to the baseline, is equivalent to 3 additional full-time persons. Thus, SEAP involves 89 full-time employees compared to 86 employees in the baseline agile development process.

Finally, a penetration tester attacks the software components that are the output from every sprint using both internal and external attacks. This is outside the current scope of SEAP, because it is handled by the QAT and does not include the development teams, or affect the risk analyzes results. Moreover, in the baseline development process, penetration tests are done for each version upgrade and are normally done by an external source.

The Application of SEAP in Software Development

In SEAP, there is an ambition of creating a security-aware culture that embraces everyone that works with the software development at a company. In this sections, a number of suggestions and hints are given to ease the introduction of SEAP into a company's already existing development process. Once the SEAP has been introduced, the overarching goal is to create an environment where all software developers should have raised their security-awareness to a level where they can make sound security-related decisions throughout development projects. This implicitly means that no single person has detailed control of every security aspect related to the product being developed, but rather that the understanding is distributed among the well-equipped and competent personnel force. Thus, SEAP provides a fast and flat sharing-culture for continuously developing the security skills among all team members.

In most cases, such a security culture does not exist to begin with, and it is not easily created unless dedication for doing so is invested. So, in order to raise the security-awareness among a company's development staff, two minimum requirements need to be fulfilled. First, there must exist at least one full-time employee at the company that has excellent core security knowledge and who is willing to share this with his/her colleagues. Secondly, there must be a number of employees that are willing to increase their knowledge in security and learn more about secure development practices. If these two requirements are fulfilled, there is a clear potential to implement the SEAP in a successful way within the company.

In addition to what has been previously explained about SEAP, it should also be added that the following three aspects that are important for successfully implementing SEAP: *internal security training*, *continuous risk analysis*, and a *virtual security group*. Each one of these are explained in more detail below.

Internal security training is important in order to raise the security-awareness among the developers. This could be managed by initiating internal workshops on a regular basis (e.g. monthly or bi-monthly), that focus on various relevant security themes, which the developers are to study in advance so that they can present their findings to each other. Such initiatives often initiate fruitful discussions even long after they took place. Another possibility is to include events like "Hack the product day(s)" once a year. During such events, external presenters are invited as

speakers, but the main aim is to let the developers hack their own product that they have worked with on a daily basis. In doing so, a double return can be provided to the company since both new and old vulnerabilities in the product could be identified. Another benefit is that the developers begin to think as attackers, which give them a mindset that allows them to better tackle security issues in future processes, e.g. by providing more useful input in prospect risk analyses.

A *continuous risk analysis* is crucial in the development of a security-aware culture, as well as, in the implementation of the SEAP method. However, it is important that the risk analyses are focused, well-structured, and carried out by the developers. To keep each risk analysis session focused, only one BUC (i.e. one requirement) should be analyzed, and the maximum timeframe for doing so should be no more than one hour. During that time, the requirement should be analyzed in thorough detail and the participants should try to identify potential vulnerabilities, which then are labelled according to their respective probability and consequence values. For each vulnerability, various safeguards, or actions should be discussed, and the most relevant and cost-effective ones should be suggested. In order to keep the risk analyses well-structured, only the developers in the development team that will implement the BUC that is being analyzed should be included. This means that the risk analysis for a certain BUC should always be carried out before it is actually implemented. The security master in the development team should moderate the session, but he/she should *not* be solving the security problems him-/herself. Instead, the security master should act as an eye-opener and a catalyst by providing feedback and asking important questions (e.g. on the form “how do you handle the aspect of ...”). In doing so, the security master can lead the identification of solutions in ways that may otherwise not have occurred to the developers. As a result, the risk analysis sessions further improve the developers’ knowledge about both general security aspects, as well as, specific issues with the product that they are developing.

The benefits from such risk analyses are typically three-fold. First, vulnerabilities in the product are identified and solutions are suggested. Secondly, that the developers increase their security-knowledge based on the discussions that occur during the risk analyses, and it also helps them get the right mindset in terms of thinking about security. Lastly, that the developers have analyzed each BUC in detail in terms of potential security issues before they actually start implementing

them. This gives them a better understanding of each BUC prior to its development. So, the risk analyses are important as they raise the security knowledge among the development teams by gradually transferring some of this knowledge from the security master to the team members. Therefore, the risk analyses are key building blocks in developing a shared security-culture among the staff.

The third aspect is the *virtual security group* that everyone with a dedicated security-related role at the company should be a member of. This security group is a web-based information group, which is why it is referred to as virtual. The virtual security group makes up an important forum in which questions, issues, and solutions related to security can be discussed. As a result of the discussions in the security group, the persons having the different roles (e.g., security masters) can keep up to date with latest developments, which further vouches for that core security knowledge and experience is preserved within the company.

Lastly, if these aspects are implemented, the general level of security knowledge will increase among the developers, which in turn is a prerequisite for creating a security-aware culture among the development teams. The goal is to build a proper security culture within the agile team by involving the full security group that fuels fast responses to upcoming situations and questions, as several people can handle them. So, SEAP involves a fast handling of security issues in a flat security-aware team with a cooperating culture involving the entire secure agile development process.

Research Method

In this study, the results from risk analyzes performed on five different versions (1.0-5.0) of the mobile money system are being compared. Version 1.0 was developed using the traditional agile software development approach used at Ericsson AB. While versions 2.0-5.0 were developed using SEAP. Thus, version 1.0 will act as the baseline in this study. The purpose of the comparison is to evaluate the different processes' ability to address risks identified during the risk analysis phase. In scientific terms the independent variable is the development method used, which has the following two levels: (1) the traditional agile process, and (2) the security-enhanced agile process (SEAP). The dependent variables are:

(1) the severity of the identified risks, (2) the proportion of risks that are corrected, (3) the proportion of risks that are postponed, and (4) the proportion of risks that are left unhandled/accepted. A repeated measures design, also known as a within-groups design was used in this study.

Our hypothesis is that both the number of risks identified and the ability to address them will improve when SEAP is used, i.e., compared to the traditional agile process. Furthermore, it is analyzed whether or not the extra resources spent on security is worth the potential improvement. A comparison of costs, in terms of full-time employees, is calculated between SEAP, i.e., product versions 2.0-5.0, and the process used during the development of version 1.0. A comparison of the risks identified in the two development processes was carried out by investigating the proportion of risks that were either (1) fixed, (2) handled as a new BUC (same as postponed), or (3) left unhandled. An assumption is also that more severe risks will be identified when SEAP is used compared to the baseline approach.

Research Questions

The following four research questions related to SEAP are investigated by comparing SEAP to the baseline version:

- RQ1. How does the use of SEAP, compared to the baseline, affect the severity of the identified risks in the developed software?
- RQ2. How does the use of SEAP, compared to the baseline, affect the proportion of corrected risks?
- RQ3. How does the use of SEAP, compared to the baseline, affect the proportion of risks that are postponed to a later software version?
- RQ4. How does the use of SEAP, compared to the baseline, affect the proportion of risks that are left unhandled?

Data

For the study in this work, risk analysis data from the development of five different versions of the mobile money transfer system are being used. The raw risk analysis data was provided in Excel format from a senior security expert at Ericsson AB. In the baseline version, a risk analysis was done once a year involving 6 to 8 persons during approximately one day. For the consecutive

versions, where SEAP was used, a risk analysis was done for every BUC, in all between 30 and 40 times during a year. Those risk analyses involved the members of each development team during approximately one hour each time.

In order to compare the risk values from the different software versions, they had to be transformed first. Since, version 1.0 of the product used a probability and consequence scale between 1-4 (inclusive). Although, no risks actually received a risk value higher than 9 in that analysis process, and no risks received a 4 in either of the two categories. Due to new routines that characterize the development of versions 2.0, 3.0, 4.0, and 5.0, the scale used for rating the probability and consequence values was extended from 1-4 to 1-5. As a result, the risk values in version 1.0 could range from 1 to 16, while they could vary from 1 to 25 in the later versions. To address this, all risk values were normalized using a unit-based transformation that brought the risk values into a scale between 0 and 1 (inclusive). The authors ensured that the transformation did not accidentally change any of the intrinsic properties of the different distributions by comparing each distribution's mean, standard deviation, skewness, and kurtosis before and after the transformations.

Statistical Tests and Data Analysis

An investigation of relevant statistical tests was made based on the characteristics of the risk data. To answer RQ1 above, a Mann-Whitney's non-parametric U-test was chosen (Sheskin, 2011). The U-test was chosen mainly because it handles discrete and non-normally distributed distributions, which applies to the collected risk data. The null hypothesis, which was tested, is that both groups are identical, i.e., that they come from the same distributions. In this study, it is therefore aspired to reject the null hypothesis, and thereby show that SEAP used at Ericsson AB while developing software versions 2.0, 3.0, 4.0, and 5.0 shows significantly improved results compared to the baseline, i.e., the development process used while developing version 1.0. Also, Cohen's d is used as a measure of effect size (the magnitude of difference) between version 1.0 and the later versions (Cohen, 1988).

In order to answer RQ2, RQ3, and RQ4, the Chi-Square two-sample test for equality of proportions **Fel! Det går inrte att hitta någon referenskälla.** was

selected. The expected proportion used by this test was the proportion of fixed, postponed, and unhandled risks of the baseline. These expected proportions were then tested against the four observed proportions for versions 2.0, 3.0, 4.0, and 5.0 of the product.

A significance level (alpha value) of 0.05 was chosen for all statistical tests in this study. All test results are presented with the corresponding test statistics and p-values, as well as standard deviations where necessary. The data is visualized using standard approaches such as tables, histograms, stacked bar charts and box-plots generated in R with the ggplot2 package.

Internal and External Validity Threats

As with any study design the current study is associated with validity threats. One internal validity threat is that we focus on risks as an indicator of the developed product's level of security. Although there are other indicators/measures of security, the authors argue that it is interesting to analyze the effect that SEAP has on for instance the number of risks identified and the risks' individual severity. Mainly, because these risks reflect the level of security in the software product and the agile team members' ability to detect them. Another validity threat that is associated with the repeated-measures design used in this study, is the so called learning effect, i.e. that the subjects improve at the studied task when it is repeated. However, such learning effects can be disregarded in this study when comparing the risk analysis data for each software version that was developed using SEAP, with the baseline version. The reason is because the risk analysis in the baseline version was handled outside the development teams, i.e. since team members had little involvement in the risk analysis for version 1.0 of the software product.

Regarding the external validity, it is the authors belief that the results in this work could be generalized to other contexts than the present one, i.e. Ericsson AB. The reason for this is because Ericsson AB was using an agile development process (as presented previously) before adopting SEAP. Thus, other organizations that also use an existing agile development process should be able to do a similar transition to SEAP. No special circumstances that would make SEAP fit particularly well at Ericsson AB have been identified by the authors.

Results

The results section is divided into six subsections that first answer the four research questions, followed by an overall risk comparison and lastly the time efficiency, i.e., average number of hours spent per risk.

RQ1: On the Effects of SEAP on Risk Severity

Figure 4. Normalized risk values for the five versions of the product, which show that more severe risks (closer to the maximum risk value of 1.0) were identified in the risk analysis process for version 2.0 to 5.0 compared to the baseline.

As is shown in Figure 4, the severity of risks for the five software versions is included. Even with only an immediate eye-examination of the content, it is revealed that the software versions that were developed using SEAP identify more

severe risks, i.e., closer to the maximum risk value of 1.0, than the baseline. The mean normalized risk values for versions 1.0-5.0 were as follows: 0.25 (sd=0.11), 0.40 (sd=0.22), 0.40 (sd=0.20), 0.46 (sd=0.20) and 0.40 (sd=0.19). In order to quantify the difference in size between the baseline and versions 2.0-5.0, the effect size using Cohen's d was calculated, which resulted in 0.82, 0.91, 1.25, and 0.96 respectively. As a rule of thumb, effect sizes larger than 0.8 are usually said to be large, while effect sizes between 0.2-0.8 are medium (Sawilowsky, 2009). It can thus be concluded that product versions 2.0-5.0, i.e., those that were developed using SEAP, identify more severe risks compared to the baseline version.

To test if any of the differences in risk severity between the baseline and versions 2.0-5.0 are statistically significant, the Mann-Whitney U-test was used. The test states that software version 2.0 identifies significantly more severe risks than the baseline ($p=5.72 \times 10^{-3}$, $W=611$). The same goes for version 3.0 ($p=7.43 \times 10^{-5}$, $W=2116$), version 4.0 ($p=7.09 \times 10^{-8}$, $W=2600$), and version 5.0 ($p=1.56 \times 10^{-4}$, $W=1736$). Based on these results, the null hypothesis can be rejected at the significance level of 0.05 and it can be concluded that SEAP identifies more severe risks than the default development process. Training effects can be disregarded because in version 1.0 these tasks were handled outside the development teams. Also, there is no improvement trend between the latter versions, i.e. an improved work process between the first version vs. later versions rather than more skilled users, which hypothetically gradually improves all five versions.

RQ2: On the Proportion of Corrected Risks

The second research question concerns the proportion of risks that are corrected, i.e., fixed so that the risk no longer impacts the software negatively, was investigated through Chi-square tests that compared these proportions for each of the versions 2.0, 3.0, and 4.0 with the baseline. The proportion and the results are presented in Table 3. The results show that the number of corrected risks for versions 2.0, 3.0, 4.0, and 5.0 are significantly higher than the number of corrected risks in the baseline with p-values of 3.19×10^{-6} , 2.59×10^{-7} , 2.26×10^{-8} , and 3.99×10^{-7} respectively. The null hypothesis is therefore rejected at the significance level 0.05, and it is concluded that SEAP corrects more risks than the default development process. The baseline corrected merely 12.5 % of the risks,

while the weighted mean (adjusted for the number of risks) for SEAP is 67.9 %, i.e. above five times more corrected risks.

Table 3. The total number of risks (N), the number of corrected risks, the number of corrected risks in %, χ^2 statistics and p-values for the three Chi-square tests between the baseline and versions 2.0, 3.0, 4.0, and 5.0 respectively.

Version	N	Corrected risks	Corrected risks in %	χ^2	p-value
Baseline	40	5	12.5	-	-
Version 2.0	21	15	71.4	21.7	3.19×10^{-6}
Version 3.0	64	41	64.1	26.5	2.59×10^{-7}
Version 4.0	64	44	68.8	31.3	2.26×10^{-8}
Version 5.0	29	21	72.4	25.7	3.99×10^{-7}

RQ3: On the Proportions of the Postponed Risks

In Table 4, the results regarding the number of risks that are postponed as they are in need of new BUCs are presented. The Chi-square tests, which compare the baseline with versions 2.0, 3.0, 4.0, and 5.0, reported the following p-values respectively 2.06×10^{-2} , 2.80×10^{-3} , 5.09×10^{-4} , and 9.38×10^{-4} . Therefore, the null hypothesis is rejected at significance level 0.05, and it is concluded that fewer risks are postponed with SEAP, by creating a new BUC request, compared to the baseline. The baseline postponed 37.5 % of the risks, while the weighted mean over version 2.0-5.0 was just 9.5 % of the risks.

Table 4. The total number of risks (N), the number of new BUCs (postponed to next version), the number of new BUCs in %, χ^2 statistics and p-values for the three Chi-square tests between the baseline and versions 2.0, 3.0, 4.0, and 5.0 respectively.

Version	N	New BUC	New BUC in %	χ^2	p-value
Baseline	40	15	37.5	-	-
Version 2.0	21	2	9.5	5.4	2.06×10^{-2}
Version 3.0	64	8	12.5	8.9	2.80×10^{-3}
Version 4.0	64	6	9.4	12.1	5.09×10^{-4}
Version 5.0	29	1	3.4	10.9	9.38×10^{-4}

RQ4: On the Proportion of the Unhandled Risks

Table 5 displays the results regarding the number of risks that are left unhandled within a particular version of the software. Unhandled risks are either accepted or left unhandled until proper fixes are identified, which then are implemented in later versions of the software. It should be noted that the unhandled risks are new

risks, not present in previous versions. The Chi-square tests, which compare the baseline with versions 2.0, 3.0, 4.0 and 5.0, reported the following p-values respectively 1.89×10^{-2} , 5.28×10^{-3} , 2.93×10^{-3} , and 2.98×10^{-2} . The null hypothesis is thus rejected at significance level 0.05, and it is concluded that fewer risks are sent back to BUC with SEAP compared to the baseline. While the baseline left 50 % of the risks unhandled, then weighted mean over version 2.0-5.0 for SEAP is merely 22.5 %, i.e., a significant decrease took place.

Table 5. The total number of risks (N), the number of unhandled risks, the number of unhandled risks in %, χ^2 statistics and p-values for the three Chi-square tests between the baseline and versions 2.0, 3.0, 4.0, and 5.0 respectively.

Version	N	Unhandled risks	Unhandled risks in %	χ^2	p-value
Baseline	40	20	50.0	-	-
Version 2.0	21	4	19.0	5.5	1.89×10^{-2}
Version 3.0	64	15	23.4	7.8	5.28×10^{-3}
Version 4.0	64	14	21.9	8.8	2.93×10^{-3}
Version 5.0	29	7	24.1	4.7	2.98×10^{-2}

A Risk Comparison Between SEAP and the Baseline Version

As can be seen in Figure 5, SEAP provides significant improvements with regards to the proportion of identified risks that are fixed. Instead of around 50 % of the risks that are accepted and left without further action as in the baseline version, SEAP contains less than 25 % accepted risks. Also, a five-fold increase of directly fixed, instead of postponed risks pushed to new sprint, risks is found. This is important since early correction of risks may avoid severe security attacks in the future. If assuming that versions 2.0-5.0 would still use the baseline development process with the same rate of accepted risks, the improvements brought by SEAP represent 78 more solved risks, i.e., solving both remaining and newly added risks. 65 % of these risks would be postponed to a new BUC in the baseline development process. The long-term quality improvement by using SEAP is related to those risks were accepted without action in the baseline, which in our case corresponds to the remaining 35 % of the risks. This implementation should also work in agile projects with fewer team members, since SEAP's main goal is to include the security competence inside the agile teams, which doesn't necessarily relate to the size of projects teams. So, in SEAP money is saved by

both solving security risks directly within the teams, as well as, improving the software product's security quality by solving a larger proportion of the risks.

Figure 5. Comparison of the proportion of accepted, postponed, and fixed/corrected risks in the baseline method (version 1.0) and the new security-enhanced agile development model, i.e., SEAP (versions 2.0, 3.0, 4.0 and 5.0).

The risks identified in the mobile money system can be further subdivided into the following four different types with regards to which part of the system they affect: the application (App), the underlying operating system (OS), the network (Net) or the database (DB). A majority of the 218 risks were related to the application, in fact, 187 of the risks (86 %) were related to the application category. As is shown in Figure 6, the risks related to the application were also slightly more severe than the other types, with a mean risk value of 0.38 (sd=0.20). The three remaining types had similar mean risk values with Net (n=14) at 0.33 (sd =0.17), OS (n=13) at 0.30 (sd =0.12) and DB (n=4) at 0.30 (sd =0.11).

Figure 6 shows the proportions of these risk types per software version of the mobile money system. As can be seen in Figure 6, the risk analysis for the baseline identified a more diverse set of risks with regards to the different types, e.g., Net, OS, and App. While the risks identified in version 2.0-5.0 are more concentrated to the application category. The reason for this difference is because risk analyses within SEAP are highly focused towards a single requirement (currently being developed) in each analysis. As these requirements, for obvious reasons, focus on the application itself, it is also logical that most risks are identified in that category. When version 1.0 was developed, a more conventional risk analysis method was

used where a single general risk analysis was carried out in the end of the project. However, it is important to note that over time, the risks analyses used in versions 2.0-5.0 identifies several risks outside the application type as well.

Figure 6. Distribution of risk values per risk type over the five software versions. The boxplot jitter visualization shows 1st quartile, 2nd quartile (i.e., the median), 3rd quartile and upper/lower fences. Risk values that extends more than 1.5 times the inter-quartile range are regarded as outliers and are shown as grey dots.

Figure 7 is similar to Figure 6 in that it shows risks per type and software version, but instead of visualizing risk value distributions, it instead shows the proportion of risks that were fixed (i.e., the source code was improved or safeguards were implemented), postponed to next sprint, or accepted (i.e., unhandled as no action was taken). Significantly more risks (both in absolute numbers and in relative proportion) were fixed in versions 2.0-5.0 compared to the baseline. This difference is large for the three types DB, Net and OS, while the difference is huge for the risks concerning the application category. All risks related to OS or database issues were fixed, while about half of the risks related to network issues were fixed (four were accepted and one postponed). Out of the risks related to the application, 68.3 % were fixed, while 10.0 % and 22.6 % were postponed and accepted respectively. For the baseline, these figures were 10.7 %, 28.6 %, and 60.7 % respectively, which is a significantly better result, in terms of improved security, when SEAP is used. Thus, it can be concluded that significantly more of the identified risks are fixed when SEAP is used, which results in an overall increase of the product's security, compared to the baseline.

Figure 7. Number of risks per risk type and software version, as well as, grouped according to how each risk was handled, i.e., either as accepted (i.e., left unhandled), postponed to next sprint, or fixed.

Efficiency in Terms of Time

The total time spent on the risk analysis tasks for each product version is shown in Table 6. Version 2.0 lasted for a short period of time so the hours spent on risk analysis are reduced. As can be seen in that table, the baseline required 2.7 employee hours spent for every risk identified in the risk analysis. As a comparison, the weighted mean value for SEAP was 1.86 hours, which is a reduction with a factor of 0.69, i.e., 31 %. Similarly, the hours spent on risk analysis tasks per corrected risk using the baseline process were 21.6 hours/risk, while the weighted mean value for SEAP was 2.56 hour/risk. This represents an impressive 8.43 times, or 88 %, reduction compared to the baseline. It should be noted that the rise in both hours per found risk, as well as, hours per corrected risk for version 5.0 probably is due to the increase in risks related to the operating system, compared to version 2.0-4.0. More OS related risks were found in version 5.0 since that version included a fully automated deployment system for the mobile money product.

Table 6. The number of hours spent on risk analysis tasks, the number of hours spent on risk analysis per identified risk, and the number of hours spent on risk analysis per corrected risk for each product version.

Version	Hours spent on risk analysis	Hours per found risk	Hours per corrected risk
Baseline	108	2.70	21.60
Version 2.0	28	1.33	0.43
Version 3.0	116	1.81	2.83
Version 4.0	84	1.31	1.91
Version 5.0	104	3.59	4.95

Discussion

In SEAP, the identification of cost-efficient methods for improving software security in the development phase is investigated. The security competences in SEAP is distributed among three different roles; security officer, security architect, and security master. An immediate observation is that the introduced security improvements are costly. Instead of one security resource available to all teams (like in more traditional agile projects), the responsibility for this task is distributed among several persons adding up to three times a bigger resource. The total cost for handling security rises from 1.2 % to 4.7 % of the total project cost, whereas the cost for the rest of the development team remained approximately the same.

To determine if this added cost, which can be seen as an investment in security, is justified, it is interesting to explore the corresponding utility side, something that could be expressed as a return on investment. In terms of analyzing and correcting the risks, there is a great added value with the new, more distributed model. More risks are corrected within the same version, and fewer risks are accepted, i.e., left unhandled. One reason for this is that security is handled within the actual development teams, rather than (as done previously) by the security officer. SEAP resulted in an impressive 8.43 times reduction in the number of hours spent on risk analysis per fixed risk.

More severe risks were identified and addressed in the risk analysis process for versions 2.0 - 5.0 compared to the baseline, the result was a more secure end-product. Introducing SEAP also conveyed a more systematic and thorough handling of the risk assessment, which most likely derives from the advanced skills in the teams and their increased awareness of the problems involved. Moreover, most risks are related to the applications rather than the categories of operating

system, network or database. With higher risk values, there is an increased tendency to send back the risk for initiating a new BUC, and the opposite for lower risk values that leave those risks unhandled. In the baseline setting, this was not possible to do.

Instead of having one general security officer for the whole project and then ten developers per team for each release (as in the baseline), SEAP has one security master and about nine developers for each development team that works on one BUC. By allocating the security masters into the development teams there is a constant presence of security in the mindset of the developers. Although the introduction of SEAP has not changed the penetration tests, it has changed the number of security test cases. Primarily due to the introduction of the security masters in the development teams, which has had the following three-fold effect:

1. As the security knowledge hub the security masters are constantly reminding developers and testers of security. They also provide a quick and easy way to ask how to test and verify a security issue without the need to book a meeting with a high-up employee or an external security team.
2. The security masters conducted their own security tests and unofficial penetration test days, usually as part of internal training events.
3. The security masters have a deep knowledge of the products since they are part of the development teams and are writing the code as well. During a penetration test the security masters are allocated to work together with penetration testers, allowing the penetration test to be effective and focused. It also has the benefit in the other direction as it works as a training session for the security masters themselves.

So, the biggest difference with SEAP is that it brings an increased focus to the risk analysis rather than the number of involved security resources. In SEAP, the focus is entirely on what happens inside a specific BUC, i.e., a more detailed level of analysis is provided. The risk analysis does not cover all security management, i.e., the involved security experts contribute also to the overall project, which consequently include both security and none security issues. As already described above, security improvements are done when:

- formulating the security aspects involving software requirements and design before the actual implementation of a new task, or

- improving secure code analyses and thus integrating security with regular software development during implementation, or
- testing the software components using both internal and external attack strategies after the implementation.

The improvements brought on by SEAP compared to the traditional agile process show that more severe risks are identified and also corrected with less hours spent on risk analysis. The costs are about ten times lower for all new versions of the product, which of course is an indication of the return on investment value mentioned above. The main reason for this improvement is that the security expertise is directly involved in the development process instead of an accessible central resource outside the project, i.e., a distributed solution is favored above a centralized. A main drawback in this study is that the results are dependent on the involved persons. For 10-20 % of the risks, there is no risk estimation available, showing the need for a coordinating function of the distributed security resources.

As in most banking applications, the direct monetary cost is not possible, or permissible, to calculate, so instead of this, an indirect argument is proposed. For each of the found risks, the company may lose money directly by criminal behaviors, such as, fraud or indirectly by loss of goodwill or reputation (this of course excludes the more regular operating errors). Lowering these risks is costlier because a new business opportunity proposal must then be initiated, possibly combined with changing priorities inside the ongoing process. In response to this, SEAP contains a two-fold improvement. First, it solves the risks immediately without the need to create new BUCs. Secondly, it contains a more efficient method inside the project development process for risk management, i.e. procedures to avoid or minimize impact of future risks. So, since all four research questions investigated resulted in positive answers, it can be concluded that SEAP shows promising results and that it should be investigated further through future research.

Conclusions and Future Work

Security is typically a small part of the costs within a software development project. In some systems, such as, those concerned with the transfer of money or other types of banking services, the need for secure software is a critically

important requirement. In this paper, an agile security-enhanced software development process, called SEAP, is proposed. SEAP increases the number of risks that are adequately addressed, while at the same time maintains the development project's security cost at a reasonable level.

The results in this work show that SEAP outperforms a baseline agile development process in an industrial setting, when it comes to security-related software development aspects in the following four ways:

- More severe risks were found because of a more detailed risk analysis focusing entirely on what happens inside a BUC, i.e., more advanced skills and a deeper awareness of the problems become available.
- The proportion of risks that were corrected within the software version under development increase more than five times, i.e., an increase from 12.5 % to 67.9 %, since more adequate security competences was included in the analysis process.
- Fewer risks were postponed to a future software version, in this case; a decrease from 37.5 % to merely 9.5 %.
- The number of unhandled risks decreased significantly from 50.0 % to 22.5 % when using SEAP.

The baseline agile software development process required 2.7 employee hours spent for every risk identified in the analysis process compared to on the average 1.86 hours in SEAP, which is a reduction by a factor of 0.69. Similarly, the hours spent on risk analysis tasks per corrected risks represents less than 12 % of the original time. So, instead of spending 21.6 hours on the average per corrected risk, approximately 2.56 hours were required when SEAP was used.

Despite the more efficient risk analysis process of SEAP, a drawback is the extra supply of resources needed for this setting. SEAP introduces three different competences, i.e., security management, security architecture, and security masters. In the baseline agile process, only a security officer is involved. The actual risk analysis that is done represents only a small part of all security work included in the SEAP setting. In fact, the daily security work by all security resources is essential for solving risks more efficiently.

Overall, a calculated 4-fold staff cost of 4.7 %, instead of the original 1.2 % out of the full project cost, is estimated for handling security in the manner prescribed by SEAP within the project. Instead of calculating the direct monetary cost (this is not possible to do in a plausible way), future work could focus on investigating the other SEAP security aspects, i.e., apart from the risk analyses, e.g. the effects it has on penetration tests or security-related test cases. Other future work could also investigate the effects of the SEAP development process in other settings than the one included in this study, i.e., the case of the development of a mobile-money transfer system at Ericsson AB.

References

- Anderson, P. (2012). Measuring the Value of Static-Analysis Tool Deployments. *IEEE Security & Privacy*, 10(3), pp. 40-47.
- Baca, D., & Carlsson, B. (2011). Agile development with security engineering activities. *Proceedings of the 2011 International Conference on Software and Systems Process* (pp. 149-158). New York: ACM.
- Baca, D., Boldt, M., Carlsson, B., & Jacobsson, A. (2015). A Novel Security-Enhanced Agile Software Development Process Applied in an Industrial Setting. *10th International Conference on Availability, Reliability and Security (ARES)* (pp. 11-19). Toulouse: IEEE.
- Bin-Othmane, L., Pelin, A., Harold, W., & Bhargava, B. (2014). Extending the Agile Development Process to Develop Acceptably Secure Software. *IEEE Transactions on Dependable and Secure Computing*, 11(6), pp. 497-509.
- Beznosov, K., & Kruchten, P. (2004). Towards agile security assurance. *Proceedings of the 2004 workshop on New security paradigms* (pp. 47-54). New York: ACM.
- Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K., & Kruchten, P. (2006). Extending XP practices to support security requirements engineering. *Proceedings of the 2006 international workshop on Software engineering for secure systems* (pp. 11-18). New York: ACM.
- Chandra, P. (2017). Software Assurance Maturity Model: <http://www.opensamm.org>, Retrived: 30 Aug 2017

- Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences*. Hillsdale, NJ: Lawrence Earlbaum Associates.
- Common Criteria. (2017). *The Common Criteria*.
<https://www.commoncriteriaportal.org/>, Retrived: 30 Aug 2017
- Cotroneo, D., & Natella, R. (2013). Fault Injection for Software Certification. *IEEE Security & Privacy*, 11(4), pp. 38-45.
- Cruzes, D., Felderer, M., Oyetoyan, D. T., Gander, M., & Pekaric, I. (2017). How is Security Testing Done in Agile Teams? A Cross-Case Analysis of Four Software Teams. *18th International Conference on Agile Processes in Software Engineering and Extreme Programming*, pp. 201-216.
- Fitzgerald, B., Stol, K.-J., O'Sullivan, R., & O'Brien, D. (2013). Scaling agile methods to regulated environments: an industry case study. *Proceedings of the 2013 International Conference on Software Engineering* (pp. 863-872). Piscataway: IEEE Press.
- Keramati, H., & Mirian-Hosseiniabadi, S.-H. (2008). Integrating software development security activities with agile methodologies. *International Conference on Computer Systems and Applications* (pp 749-754). New York: IEEE.
- McGraw, G. (2006). *Software Security: Building Security In*. Upper Saddle River, NJ: Addison-Wesley Professional.
- McHugh, M., McCaffery, F., & Coady, G. (2014). An Agile Implementation within a Medical Device Software Organisation. *International Conference on Software Process Improvement and Capability Determination* (pp. 190-201). Chambridge: Springer International Publishing.
- Microsoft. (2017). What is the Security Development Lifecycle:
<https://www.microsoft.com/en-us/sdl/default.aspx>, Retrived: 30 Aug 2017
- Murphy, B., Bird, C., Zimmermann, T., Williams, L., Nagappan, N., & Begel, A. (2013). Have Agile Techniques Been the Silver Bullet for Software Development at Microsoft? *Proceedings of the Seventh International Symposium on Empirical Software Engineering and Measurement* (pp. 1-10). ACM.
- Noopur, D. (2005). *Software Engineering Institute*. Secure Software Development Life Cycle Processes: <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=7457>, Retrived: 30 Aug 2017

- Peltier, T. R. (2010). *Information Security Risk Analysis* (3rd uppl.). Boca Raton, FL: CRC Press.
- Petersen, K., & Wohlin, C. (2009). A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software*, 82, pp. 1479-1490.
- Ramachandran, M. (2011). *Software Security Engineering: Design and Applications*. Commack, NY: Nova Science Publishers, Inc.
- Sawilowsky, S. S. (2009). New Effect Size Rules of Thumb. *Journal of Modern Applied Statistical Methods*, 8(2), pp. 597-599.
- Sheskin, D. J. (2011). *Handbook of Parametric and Nonparametric Statistical Procedures*. New York, NY: Chapman and Hall/CRC.
- Siponen, M., Baskerville, R., & Kuivalainen, T. (2005). Integrating Security into Agile Development Methods. *Proceedings of the 38th Hawaii International Conference on System Sciences* (pp. 1-7). New York: IEEE.
- Wäyrynen, J., Bodén, M., & Boström, G. (2004). Security Engineering and eXtreme Programming: An Impossible Marriage? *Proceedings of 4th Conference on Extreme Programming and Agile Methods*. 3134, pp. 117-128. Berlin: Springer Berlin Heidelberg.
- Williams, L., Meneely, A., & Shipley, G. (2010). Protection Poker: The New Software Security "Game". *IEEE Security & Privacy*, 8(3), pp. 14-20.