A Novel Security-Enhanced Agile Software Development Process Applied in an Industrial Setting

Dejan Baca[1], Martin Boldt[2], Bengt Carlsson[2], and Andreas Jacobsson[3,4]
[1]Ericsson AB, Karlskrona, Sweden, dejan.baca@ericsson.com
[2]Dept. of Computer Science and Engineering, Blekinge Institute of Technology, Karlskrona, Sweden, {mbo,bca}bth.se
[3]Dept. of Computer Science, Malmö University, Malmö, Sweden, andreas.jacobsson@mah.se
[4]Internet of Things and People Research Center, Malmö University, Malmö, Sweden

***Abstract* — A security-enhanced agile software development process, SEAP, is introduced in the development of a mobile money transfer system at Ericsson Corp. A specific characteristic of SEAP is that it includes a security group consisting of four different competences, i.e., security manager, security architect, security master and penetration tester. Another significant feature of SEAP is an integrated risk analysis process. In analyzing risks in the development of the mobile money transfer system, a general finding was that SEAP either solves risks that were previously postponed or solves a larger proportion of the risks in a timely manner. The previous software development process, i.e., the baseline process of the comparison outlined in this paper, required 2.7 employee hours spent for every risk identified in the analysis process compared to, on the average, 1.5 hours for the SEAP. The baseline development process left 50% of the risks unattended in the software version being developed, while SEAP reduced that figure to 22%. Furthermore, SEAP increased the proportion of risks that were corrected from 12.5% to 67.1%, i.e., more than a five times increment. This is important, since an early correction may avoid severe attacks in the future. The security competence in SEAP accounts for 5% of the personnel cost in the mobile money transfer system project. As a comparison, the corresponding figure, i.e., for security, was 1% in the previous development process.**

Keywords — Security, risk analysis, software development, agile method, industrial setting.

I. Introduction

In software development, agile methods have grown more and more commonplace. They are also increasingly used in the development of, e.g., web and network applications, i.e., situations where security risks are prominent characteristics. Despite these risks, most existing agile development methods have few explicit features that specifically address security. 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 (e.g., 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, a security manager that performs audits or acts as a security advisor to the developers usually handle 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 manager from the outside. Nor is it always appropriate 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 picture of how all of the software requirements are implemented, which, of course, results in difficulties to grasp yet analyze risks. Another example is the short iterations that distinguish agile development projects, which may lead to that the analysis of security flaws is overlooked or not carried out in a sufficient way. This too, is often a cumbersome activity since it ideally requires the complete picture of all software requirements. Due to the 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. In turn, this usually leads to the implementation of a fragmented "quick fix" approach to whatever security issue or risk that is identified in the iterations. Finding a way to systematically integrate security management (including risk analysis) in agile software development project while at the same time meeting all the software requirements with both quality and cost-efficiency must thus be a prioritized concern.

A common point of attention is that, when taking security into consideration, productivity may be reduced and additional development costs incurred. It is, therefore, important to predict how different ways of adding security aspects to agile processes will improve the security of the final product, so that we can make sure that any additional costs due to adding security aspects are motivated by increased product security. In that sense, there is of course also a correlation with 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.

Security-enhanced computer-based systems address issues concerned with protecting assets against internal and external threats and vulnerabilities, and thus reduce the risk of compromising them to an acceptable level. 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 or malicious practices.

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, a mobile money transfer system that handles large amounts of money has been developed. The money transfer system, which is a web-based service, intended to run as a stand-alone application on mobile units, is similar to an online banking service 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 prioritized concern.

In the performed investigation accounted for in this paper, a new security-enhanced agile software development process, hereinafter referred to as *SEAP*, is consequently introduced. Matters related to security are, in this case, handled both as a process of integrating security in the agile development method and as a way of introducing specific security tools usable for the development teams. Within agile software development, security issues need to be both time-efficient, i.e., the program code must be completed on time, and cost-efficient, i.e., the extra cost must be justified as, e.g., a positive return of investment (ROI) value. 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. Consequently, this article demonstrates how security can be integrated into an agile method deployed at Ericsson AB.

This paper is organized as follows. First, we position our SEAP against the current state of the art in the field. Then, we describe the software development process and specifically highlight the relevant constraints of our method and the setting of our studies, i.e., at Ericsson AB. Thereafter, we outline SEAP and introduce its critical tasks, 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 a traditional agile development approach) 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 is conducted, and conclusions and future work are presented.

II. Related Work

Murphy et al. [11] conducted a six-year survey at Microsoft, where they investigated the attitudes with respect to agile adoption and techniques. 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[1] or the Open Software Assurance Maturity Model[2]). 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. [10]. 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 [14]. They identified 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. [12], a method for security reassurance of software increments is proposed together with the integration of security engineering activities into an agile software development process. 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 [16]. 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 (see, e.g. Boström et al. [6]; Keramati et al. [9]; Davis [7]), which compare security engineering (SE) with agile projects, this lack of a complete overview in agile approaches makes it harder and outright prevents some common SE practices from being performed.

As stated above, it is not obvious that more time and cost-efficient code also means a 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 SE activities [16]. Although their method holds merit, they did not offer any practical experimentation to fine-tune the process.

Beznosov and Kruchten examined mismatches between security assurance techniques and agile development methods [4]. Their paper is based on literature studies and the authors identify some techniques that fit well with agile development. Wayrynen et al. [17] also conducted theoretical analyses, comparing the Common Criteria[3] to agile methods, and determined that they would benefit from more empirical data to assess their results and premises. For instance, Williams et al. [18] devised a protection poker planning game that uses developer interactions and common practices to perform risk analyses during development. In the paper, the authors perform a case study at Red Hat IT, where agile approaches are used, and evaluate their risk analysis method. Baca and Carlsson [2] proposed an agile security process in an industrial setting. Security issues were addressed using three well-known security tools (Microsoft SDL, Cigital Touchpoints[4], and Common Criteria). Test cases within imperative processes related to, e.g., requirement analysis, design, implementation, testing and release, were investigated involving people familiar with the agile project environment. One drawback with this investigation was that the security manager had an overall role in the project, i.e., it was more of an externally controlling function than a participant in the ongoing development work.

[1] http://www.microsoft.com/security/sdl/default.aspx

[2] http://www.opensamm.org/

[3] https://www.commoncriteriaportal.org/

[4] http://www.cigital.com/justice-league-blog/category/software-security-touchpoints/

Avoiding this drawback has been a main focus in the method outlined in this paper.

Moreover, safety critical computer-related failures are reported within a wide range of applications. For instance, Cotroneo and Natella [7] describe an "unintended acceleration" issue in Toyota's car fleet due to faults in the software. 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 measure in the development phase of new software, something, which has been taken into account in the design of SEAP.

III. Software Development Process

Many applications that handle large financial transactions are tempting targets for various kinds of fraud and thievery attempts. Some typical examples include prepaid telecom systems, as well as, mobile and Internet banking. These systems have millions of users and penetrating their security mechanisms can provide direct economic benefit for a (malicious) user. The large financial values are not only tempting to the 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 rights controls should be done on the server side and not in the terminals. 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 presented in this paper is Ericsson AB, 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 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 is. Furthermore, the market demands highly customized solutions, specifically due to differences in services between countries. Thus, Ericsson AB often relies on agile development methods.

A. An Agile Project

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 principles are mapped to the incremental and iterative development of SCRUM, and Extreme Programming (XP).

B. Ericsson-Specific Agile Practices

Due to the introduction of incremental and agile development methods at the company in this study, the following corporate specific practices have been introduced:

The first requirement is to have small teams 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 to testing. The result of one development project is an increase of the system of which it is intended to be a part of. Projects can be 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 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 Stable 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 focuses on the overall system where the results of the projects are integrated. When the LSV phase is completed, the system is ready to be shipped.

The anatomy plan determines the content of each LSV and the point in time when an LSV is supposed to be completed. It is based on the dependencies between parts of the system, which are produced in development 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 a release version.

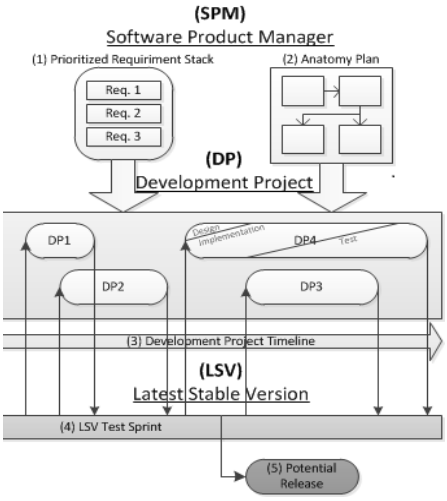

Fig. 1. An overview model of the agile processes with three distinct roles (SPM, DP and LSV).

In Fig. 1, an overview of the release process is provided. The requirements packages are created from high priority requirements stored in the repository (1). These requirement packages are implemented in projects resulting in a new increment of the product. Such a project is referred to as a Development Project (DP) that has a time-boxed duration of approximately three weeks, as (3) shows in Fig. 1. Each DP contains design, implementation and testing of the requirements. It might happen that a DP is started in one sprint (a regular, repeatable work cycle) to later be released in another. When a project has finished an increment, it is integrated with the latest version of the system, referred to as the LSV (4). The LSV has a predefined cycle and no components dropped after a new sprint will be tested until the next sprint. Based on the LSV, there can be different releases (5) of the system. These are, as stated earlier, either potential releases or customer releases.

IV. SEAP – An Extended Security-Enhanced Agile Software Development Process

As previously stated, there is one major practical drawback concerning the introduction of security in agile processes in industrial settings, i.e., the lack of a complete overview of security issues. 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 the project members do not have the necessary security skills to fix all detected bugs. The solution to this problem is to integrate security issues more closely in the agile process. So, as the agile structure is maintained, a consciousness of security is raised, as depicted in Fig. 2.

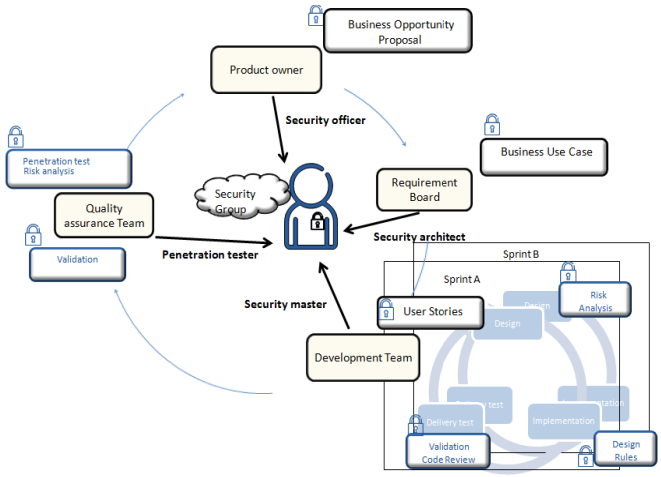

Fig. 2. The SEAP model of the agile process. A padlock indicates the occurrence of a security consideration.

A. SEAP-Activities

The product owner in conjunction with an external customer is responsible for the business opportunity proposal (BOP). 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 translates to a requirement containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it.

The development teams implement one BUC within one development project. Within the development project, all development tasks related security activities have to be performed. It is only during this project that developers are actively involved.

The security group acts as the main resource for managing security within all development teams. The group maintains different competences represented by the following four security roles.

First of all, the *security manager* handles traditional features connected to security, such as, ISO-certification, legal aspects, and other non-technical issues connected to the product owner. The security manager is responsible for the business opportunity proposal (BOP) including initiation, integration, and prioritization of requirements in the product development process.

Secondly, the *security architect* is responsible for transforming the BOP into a more technical description of the BUC by looking at, e.g., user interface, entry points into the application, the actual application, configuration, and product deployment. These requirement and design issues, i.e., the US, are then sent to development teams that take care of further design, implementation, and testing activities.

As a third role, the *security master*, is responsible for the security activities performed during the development project that implements the BUC. The development teams conduct risk analysis activities in collaboration with the security master, manage questions related to the 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.

The purpose with the risk analysis is to estimate the likelihood and degree of negative consequences of attacks to the product. The outcome is the determination of a risk value that, depending on its severity, calls for a certain degree 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 may help the developers to find and fix security flaws. In the verification phase, results of test runs 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.

A final fourth role is the quality assurance team, which handles questions related to security guidelines as part of the test analysis by using a *penetration tester*. The team has a security view deriving from the end-users perspective, meaning that the team, for instance, conduct exploratory testing and run fuzzing tools. Note that penetration testing is done from outside the project and sometimes by using external consultants. A penetration tester relies on the risk analyses already done, verifies that the system is secure by default and may use automated tools for verifying system security. Also, black box testing is utilized to find hidden or newly introduced vulnerabilities.

All security issues are handled using agile process methodology. The security group 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 case. The latter implicates that all ongoing development will be stopped until the emergency level is reduced.

B. The Risk Analysis Component

The biggest difference between the previous development process and SEAP is the risk analysis component. Traditionally, the risk analysis was conducted at the start of each major release of the product. This approach is also supported by several security-engineering practices. 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.

Security assumes that an attacker can outwit and bypass thoughtful protection and may anticipate vulnerabilities that have not yet occurred, i.e., 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 [13]). 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 headed by the security master. Each risk analysis is done during approximately one hour for a specific BUC within a sprint.

C. Risk Estimations

The probability of a successful attack has a range between 1 and 5, where 1 signifies a low likelihood of an attempt and success 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 are from the mobile money transfer product development. Each product would have to create their own ratings depending on their need. However, the ratings should always be as specific as possible so developers can easily rate the risk.

With 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 risk value, i.e., the higher the value, the greater the risk. Below a certain threshold, the team does nothing unless the cost is negligible or a synergistic effect exists, i.e., the error can be corrected in the context of a more serious correction. If the risk value is above the threshold, the threat should be corrected, resulting in an improvement in the product's security state. It may of course also be ignored if, e.g., the threat is not technically possible to solve from the local sprint group's point of view or if the cost for mitigating the threat exceeds the utility of the security fix. The threshold levels are unique for each product.

D. Actions Based on the Risk Analysis Results

A risk analysis is executed ones 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. 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 know of.

If a risk is severe, but the correction is out of scope of the BUC, then the security master will lift the risk to the security architect that creates a new BUC to implement the mitigation. The initial release might be susceptible to the risk but a future version of the product will contain corrections that prevent the risk. 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 BUC.

V. Empirical Context

The investigated project in this paper has 88 staff members distributed among 8 development teams and administration services. The teams of course have to follow Ericsson's general requirements including interacting with end-users, such as, administrators, as well as, taking care of legislative demands.

The main personnel difference between the project studied in this work and an ordinary agile project at Ericsson is the amount of resources spent on explicit security competence. Usually, there is one security manager 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 paper, we include the first version (1.0) of the studied product, which was developed according to

the Ericsson's traditional agile development methodology, i.e., the inclusion of one full time security manager, who is responsible for all security aspects.

In versions 2.0, 3.0, and 4.0 of the same product, an extended agile development process with further emphasis on security was deployed, i.e., the SEAP. The security-extended process increases the security resources to four full-time employees (see TABLE I. for further details). The main argument for making this change is the security-critical aspect of the software, i.e., as security was critical, more of the developers needed to be involved.

In this investigation, we focus on the risk analyses headed by the security masters in the development team, 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 I. THE AMOUNT OF SECURITY RESOURCES USED WITHIN THE SOFTWARE DEVELOPMENT PROCESS DURING THE DEVELOPMENT OF FOUR SEPARATE VERSIONS OF ERICSSON'S MOBILE-MONEY TRANSFER SYSTEM.

Version 1.0	Security manager	1 person á 100 %
Version 2.0 3.0 4.0	Security manager	1 person á 100 %
	Security architect	1 person á 100 %
	Security master	3 persons á 50% = 150%
	Penetration tester	1 person á 50 %

In both the traditional development process and SEAP, the security manager assists the software product manager to understand aspects about security on the product owner level, e.g., with bearing on legal and certification aspects.

In SEAP, the security architect helps the requirement board, i.e., the overall group of involved security experts, to understand aspects about security on the BUC-level, e.g., aspects concerning software security requirements or design. Also, 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 and it provides the teams easy access to a knowledgeable person regarding security aspects. Finally, a penetration tester attacks the software components that are the output from every sprint using both internal and external attacks.

VI. METHOD

In this investigation, we compare the different sets of results of the risk analysis tasks performed in the different versions, as well as, in the development processes, i.e., SEAP vs. the traditional process. The purpose is to evaluate their ability to address the risks identified during the risk analysis phase. In version 1.0, a risk analysis is done once a year involving 6 to 8 persons during approximately one day. For consecutive versions, a risk analysis is done for every BUC, in all between 30 and 40 during a year. Every risk analysis involves 3 to 4 persons during approximately an hour each time. Another difference between the two developments processes is that security experts are directly involved in the development teams in SEAP.

Our hypothesis is that both the number of risks and the ability to address them will increase when SEAP is used compared to the traditional process. Furthermore, we analyze whether the extra resources spent on security is worth the potential improvement. A comparison of costs, in terms of full-time employees, is carried out between the SEAP, i.e. product versions 2.0 to 4.0, and the traditional development process used during the development of version 1.0, which acts as the baseline in this study. A comparison of the risks identified in the two development processes was carried out by way of investigating the proportion of risks that was fixed, handled as a new BUC, or unhandled. Our hypothesis is that SEAP will identify more severe risks compared to the baseline.

The following four properties of SEAP are evaluated against the baseline:

P1. Whether the risks identified when using SEAP has statistically higher risk values compared to the baseline or not.

P2. Whether more of the identified risks are corrected when using SEAP compared to the baseline or not.

P3. Whether fewer risks are postponed to a later software version, i.e., creating a new BUC, when using SEAP compared to the baseline or not.

P4. Whether fewer of the identified risks are left unhandled when using SEAP compared to the baseline or not.

Before being able to address the first of the four properties, it is necessary to establish a way to compare the risk values between the different software versions. In version 1.0 of the product, a probability and consequence scale 1-4 (inclusive) was used. However, no risks actually got a risk value higher than 9 in that analysis process, i.e., no risks received a 4 in either of the two categories. Depending on new crew and new routines that characterize the development of versions 2.0 to 4.0, the scale used for rating the probability and consequence was extended to 1-5. As a result, the risk values in version 1.0 could range from 1 to 16, while it 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). To verify that the transformation did not change any of the intrinsic properties of the different distributions, a total of four before-and-after comparisons of each distribution's mean, standard deviation, skewness and kurtosis were made. The result showed that no change what so ever had been made to any intrinsic properties of the distributions during the transformation, i.e., only the scale was changed.

An investigation of relevant statistical tests was made based on the characteristics of the risk data. To test property P1 above, a Mann-Whitney's non-parametric U-test [15] was chosen. The U-test was chosen mainly because it handles discrete and non-normally distributed distributions, which applies to the risk values. The null hypothesis tested is that both groups are identical, i.e., comes from the same distributions. In this study, we therefore aspire to reject the null hypothesis, and thereby show that SEAP used at Ericsson AB while developing the software versions 2.0, 3.0, and 4.0 shows significantly improved results compared to the baseline, i.e., the development process used while developing version 1.0.

In order to test properties P2, P3, and P4 the Chi-Square two-sample test for equality of proportions [15] was selected. The expected proportion used by this test was the proportion of fixed, new BUC, and unhandled risks of the baseline. This expected proportion was then tested against the three observed proportions for versions 2.0, 3.0, and 4.0 of the product.

Finally, for all statistical tests in this study, an alpha value of 0.05 was chosen, which results in a significance level of 95%. For all tests that were carried out, the corresponding test statistics and p-values are presented.

VII. Results

As is shown in Fig. 3, the severity of risks (property P1) for the four software versions 1.0 to 4.0 is included. Even an eye-examination of the content reveals that the software versions developed using SEAP identifies more severe risks, i.e., closer to the maximum risk value of 1.0, than the baseline. The Mann-Whitney U-test states that software version 2.0 identifies more severe risks than the baseline ($p=7.14 \times 10^{-3}$, $W=577$). The same goes for version 3.0 ($p=1.56 \times 10^{-4}$, $W=1990$) and version 4.0 ($p=1.93 \times 10^{-7}$, $W=2453$). Based on these results, the null hypothesis is rejected at the significance level of 95% and it is possible to state that SEAP identified more severe risks than the previous development process.

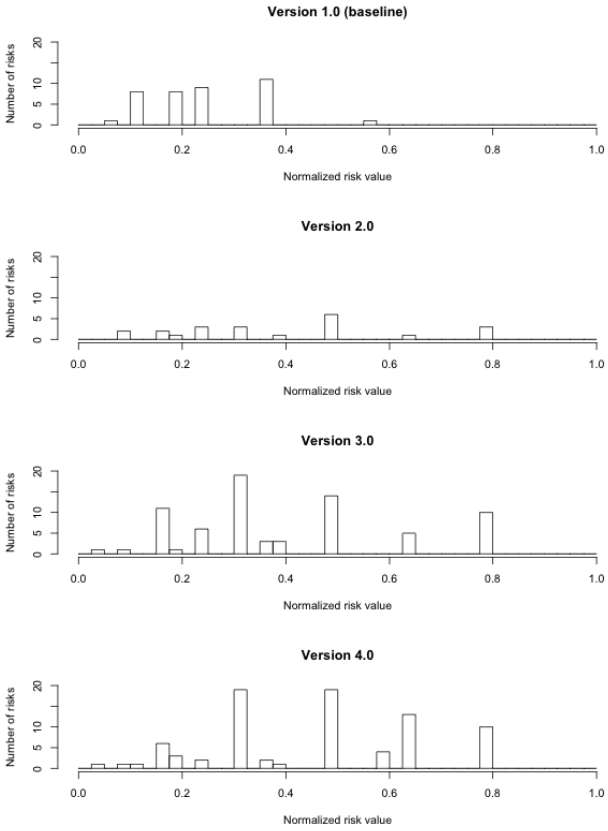

Fig. 3. Normalized risk values for the four versions of the product, which show that more severe risks (closer to the maximum risk value of 1.0) were identified in the risk analysis process for version 2.0 to 4.0 compared to the baseline.

The P2 property 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 0 The results show that the number of corrected risks for versions 2.0, 3.0, and 4.0 are significantly higher than the number of corrected risks in the baseline with the p-values of 3.19×10^{-6}, 2.59×10^{-7} and 2.26×10^{-8} respectively. The null hypothesis is therefore rejected at the significance interval of 95%, and it is concluded that SEAP corrects more risks than the previous development process.

TABLE II. The total number of risks (N), the number of corrected risks, the number of corrected risks in %, X^2 statistics and p-values for the three Chi-square tests between the baseline and versions 2.0, 3.0 and 4.0 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}

In TABLE III. the results regarding the number of risks that are in need of a new BUC, i.e. property P3, are presented. The Chi-square tests, which compare the baseline with versions 2.0, 3.0, and 4.0, reported the following p-values respectively 2.06×10^{-2}, 2.80×10^{-3} and 5.09×10^{-4}. Therefore, the null hypothesis is rejected at significance level 95%, and it is concluded that fewer risks are postponed with SEAP, by creating a new BUC request, compared to the baseline.

TABLE III. The total number of risks (N), the number of new BUCs (postsponed to next version), the number of new BUCs in %, X^2 statistics and p-values for the three Chi-square tests between the baseline and versions 2.0, 3.0 and 4.0 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}

TABLE IV. displays the results regarding the number of risks that are left unhandled within a particular version of the software, i.e., property P4. Unhandled risks are either accepted or left unhandled until proper fixes are identified, which then are implemented in later versions of the software. The Chi-square tests, which compare the baseline with versions 2.0, 3.0, and 4.0, reported the following p-values respectively 1.89×10^{-2}, 5.28×10^{-3} and 2.93×10^{-3}. The null hypothesis is thus rejected at significance interval 95%, and it is concluded that fewer risks are sent back to BUC with SEAP compared to the baseline.

TABLE IV. The total number of risks (N), the number of unhandled risks, the number of unhandled risks in %, X^2 statistics and p-values for the three Chi-square tests between the baseline and versions 2.0, 3.0 and 4.0 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}

The total time spent on the risk analysis tasks for each product version is shown in TABLE V. 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 mean for SEAP was 1.48 hours, which is almost a reduction of half of the spent time (0.55). Similarly, the hours spent on risk analysis tasks per corrected risk using the baseline process were 21.6 hours/risk, while the mean value for SEAP was 1.72 hour/risk. This represents an impressive 12.5 times reduction compared to the baseline.

TABLE V. 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

VIII. DISCUSSION

Within SEAP, the identification of cost-efficient methods for improving software security in the development phase is investigated. The security group in SEAP includes four different competences; security manager, security architect, security master and penetration tester. 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 four times a bigger resource. The total cost for handling security rises from around 1% to almost 5% of the total project cost. 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, which can to be seen 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 classified as being corrected and fewer are not handled at all. One reason for this is that security is handled within the project instead of an external group headed by the security manager, i.e., outside the development team.

More severe risks were identified in the risk analysis process for version 2.0 to 4.0 compared to the baseline. Introducing SEAP also conveyed a more systematic and thorough handling of risk assessment, which most likely derives from the advanced skills in the teams and their increased awareness of the problems involved. Instead of having one security master and 6 to 8 developers (as in the baseline case) for each release, SEAP has one security master and 3 to 4 developers for each BUC. So, the main difference is the focus on the risk analysis instead of the number of involved security resources. SEAP focuses entirely on what happens inside a BUC, i.e., a more detailed level of analysis is provided.

The risk analysis covers only a small part of all security management, i.e., the involved security experts contribute also to the overall project, which include both security and none security issues. As already described in Section V, 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 in SEAP compared to the traditional 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 discussed 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. The drawback 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.

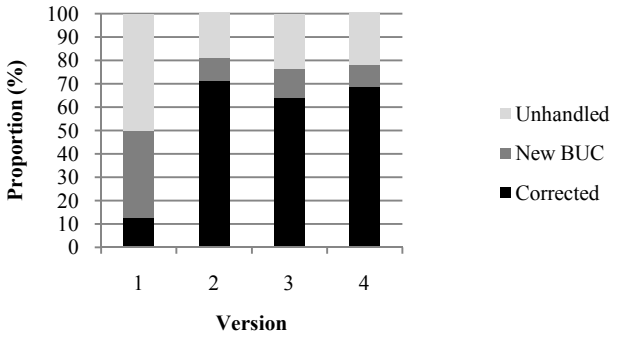

Fig. 4. The proportion of corrected risks in the original (baseline) and new security model (SEAP).

As can be seen in Fig. 4, the main findings also include improvements of the actual correction of the identified risks, i.e., risks sent back to BUC or not handled at all. Instead of around 50% of the risks left without action, SEAP has less than 25% neglected risks. Also, a five-fold increase of directly corrected, instead of sent back, risks is found. This is important since an early correction may avoid severe attacks in the future. If we postulate that versions 2.0 to 4.0 would still use the original model with the same rate of handled risks, the improvements done represent 78 solved risks. Most of these risks (51) would be handled as a new BUC in the original model. The long-term quality improvement in the model is those risks not handled at all in previous versions, which in our case corresponds to 27 risks (78-51). In all, the new security model saves money by solving the security risk directly within the team and/or improves the software product's security quality by solving a larger proportion of the risks.

As in most banking applications the direct monetary cost is not possible, or allowed, to calculate, so instead of this, an indirect argument is proposed. For each of the found risks the, company may lose money directly by fraud or indirectly by loss of reputation (this of course excludes the more regular operating errors). Correcting these risks is more costly because a new business opportunity proposal must often be initiated, possibly combined with changing priorities inside the ongoing process. In response to this, SEAP has a two-fold improvement. First of all, it solves the risks immediately without a new BUC risk analysis. Secondly, it has a more

efficient method inside the project for risk management. So, since all four properties (P1 to P4 in Section VI) we set out to investigate could be positively confirmed, we believe that SEAP shows promising results that should be investigated further through future studies.

IX. Conclusion and Future Work

Security is normally a small part of the costs within a software development project. In applications, such as, those concerned with the transfer of money or other types of banking activities, the need for secure software is a critically important requirement. In this paper, an agile security-enhanced software development process, called SEAP, is proposed that increases the number of risks that are adequately addressed, while at the same time maintaining the project's security cost at a reasonable level.

In an empirical study, we show that SEAP outperforms a traditional agile development process, when it comes to security-related aspects in the software development process, 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 in development increase more than five times, i.e., an increase from 13% to 67%, 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 38% to 11%.
- The number of unhandled risks decreased significantly from 50% to 21% when using SEAP.

The traditional agile software development process required 2.7 employee hours spent for every risk identified in the analysis process compared to on the average 1.48 hours in SEAP, which is almost a reduction of half of the spent time (0.55). Similarly, the hours spent on risk analysis tasks per corrected risk represents less than 10% of the original time, i.e., instead of spending on the average 21.6 hours, approximately 1.7 hours are spent.

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

Overall, a calculated personnel cost of 5%, instead of the original 1%, is estimated for handling security within the project. We know the number of security persons involved (5 persons) and the number of additional solved risks (78) both those with a new BUC (51) in the traditional development process and those risks not handled at all (27). Instead of calculating the direct monetary cost (this is not possible to do in a plausible way), future work should focus on investigating the security details outside the risk analysis, i.e., the SEAP development process. Also, due to the page limitation of this paper, it was not possible to describe SEAP in sufficient detail to allow its usage in other development projects. This obviously needs to be handled in future work.

References

[1] Anderson, P., "Measuring the Value of Static-Analysis Tool Deployments", Security & Privacy, Vol. 10, No. 3, pp. 40-47, 2012.

[2] Baca, D., and Carlsson, B., "Agile Development with Security Engineering Activities", Proc. of Int. Conf. on Software and Systems Process, 2011.

[3] Baca, D., Carlsson, B., Petersen, K., and Lundberg, L., "Improving Software Security with Static Automated Code Analysis in an Industry Setting", Software: Practice and Experience, Vol. 43, Issue 3, pp. 259-279, 2013.

[4] Beznosov, K., and Kruchten, P., "Towards Agile Security Assurance", Proc. of the 2004 Workshop on New Security Paradigms, 2004.

[5] Black, P.E., "Static Analyzers: Seat Belts for Your Code", Security & Privacy, Vol. 10, No. 3, pp. 48-52, 2012.

[6] Wayrynen, J., Boden, M., Beznosov, K., and Kruchten, P., "Extending XP Practices to Support Security Requirements Engineering", Proc. of the Int. Workshop on Software Engineering for Secure Systems, 2006.

[7] Cotroneo, D. and Natella, R., "Fault Injection for Software Certification", IEEE Security & Privacy, Vol. 11, No. 4, pp. 38-45, 2013.

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

[9] Keramati, H., Hassan, S., and Hosseinabadi, M., "Integrating Software Development Security Activities with Agile Methodologies", Proc. of the IEEE/ACS Int. Conf. on Computer Systems and Applications, 2008.

[10] McGraw, G., *Software Security: Building Security In*, Addison-Wesley, Upper Saddle River, NJ, 2006.

[11] Murphy, B., Bird, C., Zimmerman, T., Williams, L., Nagappan, N., and Begel, A., "Have Agile Techniques Been the Silver Bullet for Software Development at Microsoft?", Proc. of the Int. Symposium on Empirical Software Engineering, 2013.

[12] Othmane, L.B., Angin, P., Weffers, H., and Bhargava, B. "Extending the Agile Development Process to Develop Acceptably Secure Software", IEEE Transactions on Dependable and Secure Computing, Vol.11, No.6, pp. 497-509, 2014.

[13] Peltier, T.R., *Information Security Risk Analysis*, Third Ed., Taylor & Francis, 2010.

[14] Petersen, K., and Wohlin, C., "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, Vol. 82, No. 9, pp. 1479-1490, 2009.

[15] Sheskin, D. J., Handbook of Parametric and Nonparametric Statistical Procedures, Fifth Edition, Chapman and Hall/CRC, 2011.

[16] Siponen, M., Baskerville, R., and Kuivalainen, T., "Integrating Security into Agile Development Methods", Proc. of the 38th Hawaii Int. Conf. on System Science, 2005.

[17] Wayrynen, J., Boden, M., and Boström, G., "Security Engineering and eXtreme Programming: an Impossible Marriage?", Proc. of Extreme Programming and Agile Methods, 2004.

[18] Williams, L., Meneely, A., and Shipley, G., "Protecting Poker: The New Software Security Game", Security & Privacy, Vol. 8, No 3, pp. 14-20, 2010.