Security Assurance in Agile Software Development Methods: An Analysis of Scrum, XP and Kanban

Kalle Rindell¹, University of Turku
kakrind@utu.fi
Department of Future Technologies
20014 University of Turku
Finland

Sami Hyrynsalmi, Tampere University of Technology
sami.hyrynsalmi@tut.fi
Pervasive Computing
PL 300, 28101 Pori
Finland

Ville Leppänen
ville.leppanen@utu.fi
Department of Future Technologies
20014 University of Turku
Finland

ABSTRACT

Agile software development methods were introduced in the beginning of the 2000s as methods to increase the visibility and efficiency of software projects. Since then, agile methods have become a de facto industry standard in software development. However, fitting sequential security engineering development models into iterative and incremental development practices in agile methods has caused difficulties in defining, implementing and verifying the security properties of software. In addition, agile methods have also been criticized for decreased quality of documentation, resulting in decreased security assurance, necessary for regulative purposes and security measurement. As a consequence, lack of security assurance also can complicate security incident management thus increasing the software’s potential lifetime cost. This chapter clarifies the requirements for software security assurance by using an evaluation framework to analyze the compatibility of established agile security development methods: XP, Scrum and Kanban. The results show that the agile methods are not inherently incompatible with security engineering requirements.

Keywords: Security, Agile Software Development, Security Assurance, Security Engineering, Extreme Programming, Scrum, Kanban, Microsoft Security Development Lifecycle

¹ Corresponding author
INTRODUCTION

During the last decade, agile software development methods have become an industry *de facto* standard. The aim of these methods has been to improve efficiency as well as transparency of software development (Abrahamsson et al., 2002). The methods promote iterative development and informal interaction, and put a lower or even negative value to strict processes. This is particularly stressed in cases where documentation is used as a means of communication, whether used to convey the customer requirements to the development team, or for communication within the team itself, e.g., in the form of specifications (Beznosov and Kruchten, 2004; Ko et al., 2007; LaToza et al., 2006).

Introducing strict security requirements to the software development process usually results in creation of excess security assurance, such as a formal security architecture, out of necessity to fulfill the strict external security criteria. Integrating the security requirements, such as reviews, security testing, processes and documentation into an agile method, the cost of the development effort is very likely to increase (Beznosov and Kruchten, 2004). The entire extra ‘management overhead’ is in direct contradiction with agile methods’ core philosophy of leanness and informality (Beck et al., 2001). Thus, applying the security processes to the agile or lean development methods has the potential of rendering the methods, by definition, something that is neither agile nor lean.

On the other hand, the need for software security has been always one of the main drivers in software development. While quality assurance remains a key process to ensure software robustness, effectiveness and usability, security assurance provides the means to develop and deploy software components and systems that protect the system’s data, their users’ privacy and the system resources.

The operating environment of the software products and services has been evolving and changing due to extensive use of the Internet and public services as well as the ever-increasing pervasiveness and ubiquitous characteristic of software solutions. In addition, the software industry itself has gone through an unprecedented shift from sequential development methods (e.g. waterfall-type) towards iterative and incremental software development methods (e.g. agile and lean). In addition, due to the large scale adaptation of agile methods in the industry (Licorish et al. 2016, VersionOne 2018), the new agile development methods seem to be able to reclaim at least some of their claimed benefits.

Furthermore, the need for security has also been realized in the form of several commercial, international and national standards. To comply with these, several security frameworks and security-focused development methods have been presented. However, knitting together strict security engineering practices and adaptable agile software methods is not straightforward and may cause remarkable problems.

Furthermore, the selection of a software development method to be used in a development project has consequences into the software architecture and design. While the manifesto for agile software development states that the best architectures and design emerges from self-organized teams (Beck et al., 2001), this statement has been often criticized. For example, renowned software engineering researcher Philippe Kruchten (2010) has repeatedly questioned whether the concept of ‘agile architecture’ combines two incompatible approaches. In the context of security sensitive projects, this question is even more topical as it is a hard and arduous task to embed security into a product afterwards.
Therefore, the objective of this chapter is to study how well the selected agile methods are adaptable to security development practices. For the purposes of this study, we have selected three widely-used development methods, Scrum, XP and Kanban. We use Microsoft Secure Development Lifecycle (SDL) model as a benchmark for the evaluation – as the model is designed for high regulation environment and therefore its practices as well as the required frequency of occurrence should define the baseline required for this kind of activities in the industry.

These agile development methods are evaluated against a security requirement framework. The security requirement framework is created with adaption from the Finnish governmental security instructions, named VAHTI (VAHTI, 2016). The evaluation framework presented in this study consists of 22 different security requirements, selected from the instructions from software development. The compliance of the selected development methods is then evaluated against the requirements of VAHTI, and the applicability and adaptability of the ‘security-enhanced’ methods themselves are evaluated. In this analysis, SDL’s activities and their properties are used to guide the analysis.

The remaining of this chapter is structured as follows. The next section discusses shortly on different software development methods as well as related work. It is followed by a presentation of the evaluation framework and VAHTI security regulation as defined by Finnish government. After that, the results of applying the presented evaluation framework into XP, Scrum and Kanban are presented. Finally, in the last three sections the findings are discussed, future research avenues presented, and the study summarized.

BACKGROUND

The security claims of a software product or a service need to be backed with evidence; it cannot be declared only by the developer. To verify the security claims stated by the software, evidence is gathered through several activities such as reviewing the software, documentation written during the development, and processes as well as through security testing, and security auditing.

Combined, these requirements create a need for the software developers to be able to choose a development methodology that supports not only the creation of software for the selected software domain, but also satisfies the security requirements (SAFECode 2012). Preferably, this is done in the most efficient way possible, taking into account the organization and the operational environment. In software development, efficiency is gained by close integration to the utilized development methodology.

In the following, we will first take a look on different agile software development methods later studied in this chapter. It is followed by a review of related studies done in examining security engineering in agile software development context.

Software development methods

A series of different kinds of software development methods have been presented (Abrahamsson et al., 2002). In our use, a ‘software development method’ defines how a software development process is divided into different phases, how the phases are arranged, and what kinds of artefacts are produced during the different phases. Instead of more traditional methods, as so-called ‘waterfall’ and spiral models, our focus turns on modern lightweight development methods labelled as agile.
For this study, we have chosen three agile software development methods: Extreme Programming (XP), Scrum, and Kanban. These methods were selected as they are among the most popular and most used ones in the latest surveys (Licorish et al. 2016) as well as they have been the among the most popular ones during the last decade (cf. VersionOne 2013, 2018). That is, these methods can be interpreted to present the core of agile software development methods. All of these methods can be considered archetypes of agile software development methods, with strong use base in the industry. Thus, they are also eligible candidates for development work carried out in highly regulated environments.

In addition to the selected agile methods, we take a look on Microsoft’s SDL model that acts as a reference point for the evaluation. As the SDL model is designed especially for highly regulated information security environments, it should lay the needed level of security engineering activities in software projects.

**Extreme Programming.** XP is one of the first and most widely used agile development models. The XP method principally consists of a collection of practices and values; that is, it does not define strictly how the actual development process should be carried out (Beck, 1999). In addition, XP promotes a number of techniques, such as 40-hours week, iteration planning game, test-first development and small releases.

However, the guidelines given by the method are quite practical, such as the use of pair programming or continuous integration. Popularity of the XP method, especially in the beginning of the first decade of the 21st century, has spun attempts to bring security elements into the method. Previous work of security enhancements into the XP method consist of security-related user stories and abuser stories in the planning phase (Boström et al., 2006; Ge et al., 2007).

**Scrum.** Scrum can be considered as the current mainstream of the software industry (see VersionOne, 2018). The method defines certain roles for the team members, their responsibilities as well as certain tools, activities and a loose framework for the development process. The development work is divided into sprints that usually last from two to four weeks. (Schwaber, 1995, 2004)

In the extant literature, earlier examples of security enhancement to Scrum consist of loosely SDL-based security features specifically aimed for regulated environments, such as in a case presented by Fitzgerald et al. (2013). These features and processes include ‘hardening sprints’, which consist entirely of security-related planning and documentation, and regulation-driven security and quality assurance reviews and checks. This methodology includes new roles that are not included in baseline Scrum. Scrum has been selected due to its overwhelming popularity in the current software development industry (VersionOne, 2013, 2018).

**Kanban.** Kanban, much like XP, can be understood simply as a set of development concepts, ideas and principles, rather than a tightly-defined set of processes, practices and roles. It therefore provides a rather flexible framework for development, focusing on the work flow: the work flow is visualized, the amount of work in progress is limited, and the development lead time is measured. This helps the developers to stay informed of the work backlog, aims to ensure that the team members do not get overloaded, and provides metrics to optimize the development time. Kanban is typically combined with more prescriptive methods, leading into creation of e.g. Scrumban (Nikitina et al., 2012) and other hybrid methods.
Security Development Lifecycle. Microsoft’s effort to improve the security of their software has led them to develop their own security framework, the Security Development Lifecycle process. SDL is based on iterative spiral model borrowed from and adaptable to agile methodologies (Microsoft, 2012; Baca and Carlsson, 2011).

The approach selected in the SDL, when adapted without modification, is quite heavy on processes, documentation and organization – a contrast to the agile methods discussed above. This forms part of the motivation of this study, aiming to identify the minimal set of SDL elements required to fulfill the security requirements. SDL divides activities into three categories: one-time requirements, bucket requirements and every-sprint requirement. The naming of the categories appears to suggest that SDL for Agile is meant for Scrum or Scrum-like methods. SDL emphasizes the use source code static analysis tools as the principal security activity, followed by threat modeling and security testing.

Related Work
A starting point for the research was formal categorization of security aspects for software development, and conducting a feature analysis using DESMET (Kitchenham et al., 1997). DESMET framework was presented in the 1990s for evaluation of software development methods. The evaluation may be quantitative or qualitative, and based on experiments, case studies, screenings, effect analyses, feature analyses or surveys. The nature of this study suggested a screening feature analysis, with easily quantitative results: the requirement is either fulfilled or not, and each method and security process is analyzed as a simulated case study, based on expert opinions and without instantiation.

Beznosov and Kruchten (2004) earlier used a similar approach; however, they did not use established security criteria or framework, nor an external evaluation criteria. In addition, there exists a number of studies concerning secure software development concept in general, also covering the topic of security-focused testing (Fitzgerald and Stol, 2014). Furthermore, Abrahamsson et al. (2003) made an early contribution comparing agile software development methods by their suitability for different stages of development life cycle, and their support for project management — both important aspects from the security point of view. In their study, security engineering was not considered per se, as they focused more on general issues in different agile software development methods.

Regarding more specifically literature on security engineering in agile software development, a series of work has been presented (cf. Rindell et al., 2017b). A common nominator for the earlier literature seems to be an approach of documenting a proprietary corporate software development method, or even specify own development method (e.g., Baca and Carlsson, 2011; Boström et al., 2006; Rindell et al., 2015). For example, Baca and Carlsson (2011) also compare existing software methodologies, including SDL, with a proprietary method, and claim also a new proprietary method of their own. In addition, for instance Vuori (2011) discusses how IEC 61508 standard can be used to improve agile software development methods’ suitability for a safety-critical development environment.

Furthermore, Wäyrynen et al. (2004) discusses whether security engineering approach can be implemented into XP development method at all. They also discussed certain activities – e.g., static code reviews, security policies – that might need to be included into XP to make it more security engineering friendly. Boström et al. (2006) continued this topic and proposed the use of
abuser stories and security-related user stories as a part of XP to make it more suitable for security engineering.

Similar approach has been utilized by Köngsli et al. (2006), who notes the mismatch between requirements for security engineering projects and agile software development. Yet, the study does not specifically discuss on the claimed shortcomings. Nevertheless, the study reviews a bulk of literature and lists several different security activities that should be taken into account. Chivers et al. (2005) also note the mismatch between XP and security engineering. Their proposal is iterative security architecting to maintain a ‘good enough’ (from a security point-of-view) architecture during iterative and incremental development work. Also Ge et al. (2007) aim to improve security engineering in XP through security training and security architecture. Their security activities include, e.g., security stories.

Adelyar and Norta (2016) and Adelyar (2018) adopted a different approach and studied what kind of security challenges there are in agile methods. Their results show that there are certain developer and customer activities that might enable the presence of security flaws in the software product.

De Win et al. (2009) used an alternative approach and focused on three security engineering standard processes in the field: CLASP, SDL and Touchpoints. Their focus is on these practices and whereas they acknowledge, e.g., XP, it is not clear how well the studied cases are suitable for mainstream software security engineering projects with an agile approach. Ayalew et al. (2013) continue this work and focus on cost-benefit analysis. However, their work produces also a list of comparable agile security activities.

Sonia et al. (2014) assigns an agility value to different security engineering activities. In their analysis, all security activities studied are seem to be incompatible with at least one modern agile activity. Thus, their work further emphasizes the difference and incompatibility of security engineering and agile software development methods.

Othmane et al. (2014) propose their own secure agile development method for continuous security. Their approach is to adapt different security engineering activities in each development sprint in order to guarantee that the software produced in each iteration is secured.

In addition, empiric research has been conducted on the impact of agile methods on software security (Alnatheer et al., 2010). There also exists case studies in adapting Scrum-based methods in an environment with strict formal security regulations (Fitzgerald et al., 2013; Rindell et al., 2016, Oyetoyan et al., 2016).

The study by Fitzgerald et al. (2013) discusses security regulations and Scrum in considerable depth, yet only within the scope of a single company and a single development method. The case study by Rindell et al. (2016, 2017a) focuses also on a case using Scrum-based method but it is as well restricted by the scope of a single project. Oyetoyan et al. (2016) studied two organizations using Scrum in their development work. The focus of the study is on the skills, trainings and experience of the development teams. The study shows that security awareness improves the use of security activities in the development work.

To summarize the review of related work, it can be noted that there are a series of work devoted for improving different aspects of security engineering in agile software development methods. However, a common claim seems to be that agile methods are inherently insecure and
inapplicable for security engineering (Rindell et al., 2017b). Yet, there is a lack of work addressing whether this can be considered to be true or just a myth.

This study takes an alternative approach and aims to evaluate the vanilla versions of different agile software development methods against a security engineering framework. To depart from the previous work, that have assumed the insecurity of these methods, we instead use an industry standard (i.e., VAHTI regulation) as a baseline for defining security activities needed.

**EVALUATION FRAMEWORK**

This chapter aims to assess how well agile software development methods are suitable for security engineering projects by using a nationwide criteria and new development methods, and the SDL security framework as a benchmark. In contrast to most of the previous approaches, security is in this chapter considered to be an absolute requirement for software. In the agile method terminology, security is considered an essential part of the customer satisfaction, which the agile methods aim to promote (Beznosov and Kruchten, 2004).

We concentrate on the challenges this brings into the development process and the quality assurance closely associated with security controls. This study also makes the security requirement more specific by using a well-established security criteria, VAHTI, and inspecting the applicability of the selected software methodologies to comply with this criteria. The term ‘security assurance’ is used to describe the drive to mitigate security vulnerabilities in a measurable and evidence-backed way, comparable to the term ‘quality assurance’, which aims to mitigate software discrepancies in general. In a regulatory context, security assurance conveys also the meaning of security proof, referring to written security documentation and e.g. logs.

The specific set of security requirements used in this chapter is based on VAHTI (literally translated ‘Guard’, a mnemonic acronym in Finnish for ‘Government Information Security’) security regulation. VAHTI is one of the earliest and most comprehensive sets of open and public information system security regulations. The instructions consist of 51 documents, published since 2001, aiming at covering the whole life cycle of the governmental information systems. The guideline covers also various aspects of information systems management, governance, use, and, ultimately aid implementing Finland’s national information security strategy, published in 2009.

VAHTI instructions specify three security levels (‘basic’, ‘increased’ and ‘high’). The instructions were originally targeted only for government’s internal information systems work; due to public sector’s integral part in the Finnish society, VAHTI is in the process of becoming a de facto security standard in any information system that interacts with a governmental system. VAHTI exists to harmonize the security requirements among the public institutions, a set of national standards has been developed, based on standards such as ISO/IEC 27002 (ISO/IEC, 2013), and derived from Systems Security Engineering – Capability Maturity Model (ISO/IEC standard 21827, 2008). These requirements aim to cover the life cycle and various use cases of the public information systems, and span over several dozen public documents.

VAHTI was selected due to two main reasons. First, the selection of a governmental regulation that is in active use in Finland allows us to focus on aspects and activities that are actively used in the industry. That is, instead of developing an own evaluation framework from the scratch, we depart from the previous work by using an existing and widely-used instruction set in our analysis. Second, VAHTI allows us to focus on a more comprehensive picture of a security
engineering instead of a single standard. While this also forces us to select the suitable criteria for the evaluation of the software development methods, the evaluation framework based on this kind of an instruction set should be more general than one based on a more narrow standard.

For developing the evaluation framework for this study, we focused on all VAHTI criteria in different phases and regulation levels. VAHTI security criteria for application development comprises the whole life cycle of software. The complete list of security requirements in VAHTI includes 118 activities (FMoF, 2013). This list was analyzed through by the authors and relevant criteria for the software development methods were selected. The selection was made based on the relevance of the requirements to the software development method: the selected criteria are either requirements for the documentation, reviews or the development process itself.

We excluded requirements that did not directly relate to the software development method or approach used. For example, requirements for handling data storage as well as physical access to the server room are not relevant for the software development method’s point-of-view. That is, those can be handled regardless of the development method used.

The only selected organizational requirement, the one for security training, was included due to the fact that SDL as well as previous work (e.g., Ge et al., 2007 and Oyetoyan et al., 2016) emphasizes this as a security-enhancing mechanism. Furthermore, it may affect the development roles in the software development method.

From the complete list of VAHTI’s security requirements, 22 activities directly concerning the software development methods were selected. The selected security requirements are presented in Table 1. The table also links each requirement back to the respective VAHTI security level.

We evaluate each criteria through four dimensions: i) requirements integrality in a development method, ii) requirement’s frequency of occurrence in the development work, iii) level of automation and iv) cost of performing or carrying out the requirement. Each of these dimensions are clarified more in the latter.

In the evaluation, we study whether each requirement is i) an integral part of the method; ii) the method can be adjusted to support the requirement; or iii) the method is incompatible with the requirement or there are needs for improvement in the method. This evaluation was done by the authors discussing and analyzing how well the requirement can be implemented in each of the selected software development methods. All authors are expert in software engineering and it was required that a mutual agreement needed to be reached before continuing.

The other assessment criteria of the frequency requirement for each technique is defined as it is by SDL. That is, the analysis of these requirements needed frequency is based on how often SDL requires them to be used. The frequency is encoded into three values: i) one-time requirements; ii) bucket requirements and iii) every-sprint requirements.

Each task is further ranked to either Automated, Semi-automated or Manual. Semi-automated means the bulk of the work is done by automated tools, which in turn may require a considerable amount of manual configuration. This analysis is based on the description in the VAHTI instruction set and it is subjectively evaluated by the authors. However, the authors mutually agreed on the level of the automation during the analysis.

Cost is calculated by multiplying the level of automation with the frequency of the task. The cost of security requirement’s, that has to be carried out only one-time during the project and which can be easily automated, would be considered as a very low. Similarly, manual requirement that
has to be carried out in each sprint would be considered as a very high in its cost. The remaining combinations are similarly categorized into the following scale: Very Low, Low, Medium, High, and Very High.

Table 1. The evaluation framework’s criteria

<table>
<thead>
<tr>
<th>No.</th>
<th>Criterion</th>
<th>VAHTI level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Application Risk Analysis</td>
<td>Basic</td>
</tr>
<tr>
<td>2</td>
<td>Test Plan Review</td>
<td>Basic</td>
</tr>
<tr>
<td>3</td>
<td>Threat Modeling</td>
<td>Basic</td>
</tr>
<tr>
<td>4</td>
<td>Goal and criticality</td>
<td>Increased</td>
</tr>
<tr>
<td>5</td>
<td>Business Impact Analysis</td>
<td>Increased</td>
</tr>
<tr>
<td>6</td>
<td>Documentation of Security Solutions</td>
<td>Increased</td>
</tr>
<tr>
<td>7</td>
<td>Application Security Requirements</td>
<td>Increased</td>
</tr>
<tr>
<td>8</td>
<td>Application Security Settings Definition</td>
<td>Increased</td>
</tr>
<tr>
<td>9</td>
<td>Security Testing</td>
<td>Increased</td>
</tr>
<tr>
<td>10</td>
<td>Security Auditing</td>
<td>Increased</td>
</tr>
<tr>
<td>11</td>
<td>Architecture guidelines</td>
<td>High</td>
</tr>
<tr>
<td>12</td>
<td>External Interface Review</td>
<td>High</td>
</tr>
<tr>
<td>13</td>
<td>Use of Secure Design Patterns</td>
<td>High</td>
</tr>
<tr>
<td>14</td>
<td>Attack Surface Reduction</td>
<td>High</td>
</tr>
<tr>
<td>15</td>
<td>Architectural Security Requirements</td>
<td>High</td>
</tr>
<tr>
<td>16</td>
<td>Internal Communication Security</td>
<td>High</td>
</tr>
<tr>
<td>17</td>
<td>Security Test Cases Definition</td>
<td>High</td>
</tr>
<tr>
<td>18</td>
<td>Test Phase Code Review</td>
<td>High</td>
</tr>
<tr>
<td>19</td>
<td>Use of Automated Testing Tools</td>
<td>High</td>
</tr>
<tr>
<td>20</td>
<td>Security Mechanism Review</td>
<td>High</td>
</tr>
<tr>
<td>21</td>
<td>Application Development-time Auditing</td>
<td>High</td>
</tr>
<tr>
<td>22</td>
<td>Security training</td>
<td>High</td>
</tr>
</tbody>
</table>

RESULTS

Table 2 summarizes the Scrum’s, XP’s and Kanban’s compliance with each security requirement in the evaluation framework. In addition, Table 3 reports each criterion’s level of automation and estimated cost as this was found an important factor. In the following, we will discuss on the central observations based on these analyses.

Table 2. Compliance of XP, Scrum and Kanban with different security requirements presented in the evaluation framework

<table>
<thead>
<tr>
<th>No.</th>
<th>Requirement</th>
<th>XP</th>
<th>Scrum.</th>
<th>Kanban</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Application Risk Analysis</td>
<td>Integral</td>
<td>Integral</td>
<td>Integral</td>
</tr>
<tr>
<td>2</td>
<td>Test Plan Review</td>
<td>Adaptable</td>
<td>Adaptable</td>
<td>Adaptable</td>
</tr>
<tr>
<td>3</td>
<td>Threat Modeling</td>
<td>Integral</td>
<td>Integral</td>
<td>Integral</td>
</tr>
<tr>
<td>4</td>
<td>Goal and Criticality</td>
<td>Adaptable</td>
<td>Adaptable</td>
<td>Adaptable</td>
</tr>
<tr>
<td>5</td>
<td>Business Impact Analysis</td>
<td>Adaptable</td>
<td>Adaptable</td>
<td>Adaptable</td>
</tr>
<tr>
<td></td>
<td>Documentation of Security Solutions</td>
<td>Adaptable</td>
<td>Adaptable</td>
<td>Adaptable</td>
</tr>
<tr>
<td>---</td>
<td>-------------------------------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>7</td>
<td>Application Security Requirements</td>
<td>Integral</td>
<td>Integral</td>
<td>Integral</td>
</tr>
<tr>
<td>8</td>
<td>Application Security Settings Definition</td>
<td>Adaptable</td>
<td>Adaptable</td>
<td>Adaptable</td>
</tr>
<tr>
<td>9</td>
<td>Security Testing</td>
<td>Integral</td>
<td>Integral</td>
<td>Integral</td>
</tr>
<tr>
<td>10</td>
<td>Security Auditing</td>
<td>Adaptable</td>
<td>Adaptable</td>
<td>Adaptable</td>
</tr>
<tr>
<td>11</td>
<td>Architecture guidelines</td>
<td>Adaptable</td>
<td>Adaptable</td>
<td>Adaptable</td>
</tr>
<tr>
<td>12</td>
<td>External Interface Review</td>
<td>Integral</td>
<td>Integral</td>
<td>Integral</td>
</tr>
<tr>
<td>13</td>
<td>Use of Secure Design Patterns</td>
<td>Integral</td>
<td>Integral</td>
<td>Integral</td>
</tr>
<tr>
<td>14</td>
<td>Attack Surface Reduction</td>
<td>Integral</td>
<td>Integral</td>
<td>Integral</td>
</tr>
<tr>
<td>15</td>
<td>Architectural Security Requirements</td>
<td>Integral</td>
<td>Integral</td>
<td>Integral</td>
</tr>
<tr>
<td>16</td>
<td>Internal Communication Security</td>
<td>Integral</td>
<td>Integral</td>
<td>Integral</td>
</tr>
<tr>
<td>17</td>
<td>Security Test Cases Definition</td>
<td>Adaptable</td>
<td>Adaptable</td>
<td>Adaptable</td>
</tr>
<tr>
<td>18</td>
<td>Test Phase Code Review</td>
<td>Integral</td>
<td>Integral</td>
<td>Integral</td>
</tr>
<tr>
<td>19</td>
<td>Use of Automated Testing Tools</td>
<td>Integral</td>
<td>Integral</td>
<td>Integral</td>
</tr>
<tr>
<td>20</td>
<td>Security Mechanism Review</td>
<td>Adaptable</td>
<td>Adaptable</td>
<td>Adaptable</td>
</tr>
<tr>
<td>21</td>
<td>Application Development-time Auditing</td>
<td>Incompatible</td>
<td>Incompatible</td>
<td>Incompatible</td>
</tr>
<tr>
<td>22</td>
<td>Security training</td>
<td>Adaptable</td>
<td>Adaptable</td>
<td>Adaptable</td>
</tr>
</tbody>
</table>

The three first criteria (Application Risk Analysis, Test Plan Review and Threat Modeling) of the framework can be considered as a basic requirement for all kinds of application development projects. That is, Application Risk Analysis is an essential security element and it has been or it can be well integrated into all methods. Yet, it can be done only manually as SDL requires it to be carried out in each iteration. Therefore, it is costly.

Test Plan Review is an internal security activity in which the personnel reviews the test plan. All of the studied development methods support this integrally while its cost is high. Threat Modeling consists of compiling the list of the threats and keeping that up to date during every sprint. This is a cornerstone activity of SDL and essential to any security related development project. The activity provides a baseline for risk analysis and guides architectural choices, among other things. The threat ‘landscape’ is dependent on the software’s intended users and use environment. This is essential to all methods and easily integrated to them. This requirement was not mandatory even at VAHTI’s highest level, which can be considered as a clear omission to the instruction set. A potential explanation to this could be the restrictions in the availability of threat modeling tools. Alternatively threat modeling can be performed as a manual task using e.g. Microsoft’s STRIDE mnemonics for component analysis. In this approach, the system and its components are reviewed for security threats in the categories of Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of Privilege.
Table 3. The level of automation, frequency of occurrence and relative cost of the evaluation framework’s security requirements

<table>
<thead>
<tr>
<th>No.</th>
<th>Requirement</th>
<th>Level of automation</th>
<th>Frequency</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Application Risk Analysis</td>
<td>Manual</td>
<td>Every sprint</td>
<td>Very high</td>
</tr>
<tr>
<td>2</td>
<td>Test Plan Review</td>
<td>Manual</td>
<td>Bucket</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>Threat Modeling</td>
<td>Manual</td>
<td>Every sprint</td>
<td>High</td>
</tr>
<tr>
<td>4</td>
<td>Goal and Criticality</td>
<td>Manual</td>
<td>One-time</td>
<td>Low</td>
</tr>
<tr>
<td>5</td>
<td>Business Impact Analysis</td>
<td>Manual</td>
<td>Bucket</td>
<td>Low</td>
</tr>
<tr>
<td>6</td>
<td>Documentation of Security Solutions</td>
<td>Manual</td>
<td>Bucket</td>
<td>Low</td>
</tr>
<tr>
<td>7</td>
<td>Application Security Requirements</td>
<td>Manual</td>
<td>Bucket</td>
<td>High</td>
</tr>
<tr>
<td>8</td>
<td>Application Security Settings Definition</td>
<td>Manual</td>
<td>Bucket</td>
<td>High</td>
</tr>
<tr>
<td>9</td>
<td>Security Testing</td>
<td>Automated</td>
<td>Every sprint</td>
<td>Medium</td>
</tr>
<tr>
<td>10</td>
<td>Security Auditing</td>
<td>Manual</td>
<td>One-time</td>
<td>Low</td>
</tr>
<tr>
<td>11</td>
<td>Architecture guidelines</td>
<td>Manual</td>
<td>One-time</td>
<td>Low</td>
</tr>
<tr>
<td>12</td>
<td>External Interface Review</td>
<td>Manual</td>
<td>Bucket</td>
<td>High</td>
</tr>
<tr>
<td>13</td>
<td>Use of Secure Design Patterns</td>
<td>Manual</td>
<td>One-time</td>
<td>Low</td>
</tr>
<tr>
<td>14</td>
<td>Attack Surface Reduction</td>
<td>Manual</td>
<td>Every sprint</td>
<td>Very high</td>
</tr>
<tr>
<td>15</td>
<td>Architectural Security Requirements</td>
<td>Manual</td>
<td>One-time</td>
<td>Low</td>
</tr>
<tr>
<td>16</td>
<td>Internal Communication Security</td>
<td>Semi-automated</td>
<td>Every sprint</td>
<td>Medium</td>
</tr>
<tr>
<td>17</td>
<td>Security Test Cases Definition</td>
<td>Manual</td>
<td>Every sprint</td>
<td>Very high</td>
</tr>
<tr>
<td>18</td>
<td>Test Phase Code Review</td>
<td>Manual</td>
<td>Every sprint</td>
<td>Very high</td>
</tr>
<tr>
<td>19</td>
<td>Use of Automated Testing Tools</td>
<td>Automated</td>
<td>Every sprint</td>
<td>Very low</td>
</tr>
<tr>
<td>20</td>
<td>Security Mechanism Review</td>
<td>Manual</td>
<td>One-time</td>
<td>Low</td>
</tr>
<tr>
<td>21</td>
<td>Application Development-time Auditing</td>
<td>Manual</td>
<td>One-time</td>
<td>Low</td>
</tr>
<tr>
<td>22</td>
<td>Security training</td>
<td>Manual</td>
<td>Every sprint</td>
<td>Very high</td>
</tr>
</tbody>
</table>

The following six criteria (i.e., the requirements from number 4 to 10 in Table 1) can be considered to be essential to all security engineering projects with increased information security alertness. With the notable exceptions of Application Security Requirements and Application...
Security Settings Definition, the cost of these requirements is estimated to be low in a development project.

The Goal and Criticality requirement means classification of the software and documentation of its purpose. Both XP and Scrum were found lacking in this respect while the Scrum-based methods are more readily adaptable to produce planning phase documentation. Kanban-based methods are also considered adaptable.

Business Impact Analysis is basically method independent requirement, and as such, considered adaptable to all methods. This document should be produced in the planning phase, and updated during the implementation when the application’s incremental threat analyses implicate further threats to the business environment.

Documentation of Security Solutions is a direct requirement to communicate the security requirements to the developers through documentation. All agile methods are fundamentally against this approach, and will need improvement to be able to take into efficient use.

Document on Application Security Requirements is a high-level description, covering the criticality of the information handled by the software, threat analysis, and other functional security requirements. All security related development methods were deemed to support creation of this document in the planning phase. Similarly, Application Security Settings Definition is an extensive documentation step, where all the software settings, interfaces, administration steps, test data, encryption details etc. are listed and thoroughly documented. A suggested action would be a separate documentation sprint, to be added into the agile methods.

On one hand, Security Testing states that security testing should be incorporated into the standard testing procedure. This requirement is supported by all methods. On the other hand, Security Auditing is a requirement for Increased and High VAHTI levels. Furthermore, it requires an external auditor. This requirement was included due to its strain on the development process, mainly through architecture auditing. Also this requirement is supported by all selected methods.

The remaining twelve criteria can be considered to be essential for a software development project where information security requirements are extremely high. As the VAHTI instruction set is defined by a governmental agency, it is not a surprise that special emphasizes has been put on these requirements. Furthermore, on average these requirements are more costly to carry out than the ones belonging essentially in the increased security category.

Architecture guidelines define the principles guiding the application development, in this context especially from the security point of view. This requirement is adaptable to all development methods. External Interface Review is an analysis of the software’s external interfaces and comparison to architectural and application level principles. All methods support the performance of this action. Use of Secure Design Patterns mandates classifying the software due to its architecture type, such as client-server, mobile, web or embedded application. The design pattern is then selected based on the architectural type. All of the studied methods support this requirement.

Attack Surface Reduction includes identifying and analyzing all software functionality where the participants cannot completely trust each other, such as open services, user or administrator actions or database connections. All methods support this step. Architectural Security Requirements mean analysis of the application’s architecture against known or anticipated threats. All methods support this requirement. Internal Communication Security concerns
especially applications utilizing multi-tier architecture and ties the deployment of the application into the development phase. Largely method independent planning-phase activity, but still supported by each method.

**Security Test Cases Definition** is an absolute requirement for almost all security-related development, and VAHTI gives here specific instructions how the test cases should be defined, such as use of empiric evidence, known issues and several sources. This requirement is adaptable to all methods. **Test Phase Code Review** is informally performed by the internal security personnel, and documented either separately or even straight into the source code. This requirement is also supported by all of the studied methods.

**Use of Automated Testing Tools** is more or less standard practice for all agile software development, regardless of the used methodology. On the security side, the tools include fuzz testing tools, vulnerability scanners, code analyzers and continuous integration tools. This requirement is quite naturally supported by all three included agile methods.

**Security Mechanism Review** is a code-level review of how security components are implemented. Basically it is method independent, but may be difficult to implement in iterative methods as after changes this review has to be done again. As such, this activity might require a specific hardening sprint, as a time-consuming activity like this may be difficult to fit into the work flow.

**Application Development-time Auditing** is a high-level security audition at various points of application development. Intrinsically a waterfall-type approach, causing difficulties with iterative methods. **Security training** means organizing purpose-oriented and role-based training for the personnel responsible for the application development, such as the product owner, developers and testers. This requirement is adaptable to all of the studied methods.

**DISCUSSION**

The three agile methods (Scrum, XP and Kanban) studied in this chapter were found to have certain issues with adaptability of security tasks. Repetitive (i.e., multi-sprint or every-sprint) documentation and review tasks were found specifically incompatible with these methods. While a theoretical examination cannot establish concrete benefits gained from the use of agile methods, it was deemed unjustifiable to claim that their use would have an adverse effect on security assurance. Instead, use of agile prompts including security items e.g. in the user stories and integrating them into the backlog as a part of regular conduct of an agile development project. All three methods were found inherently compatible with or adaptable to all planning and implementation phase activities. Incorporating security reviews and auditing into the iterative development processes proved to be a tougher issue. The SDL prompts these activities to be completed in “each and every sprint”, or “the sprint is deemed incomplete”. However, the wording of the SDL has since 2015 been altered, and the word **sprint** is no longer used as a synonym for **release**, which is the current term. It appears that Microsoft, too, has awoken to the reality in which not every agile sprint produces a released version of software.

When compared to the previous work in the field, this study is produces somewhat different results, explained by the different research approach. Whereas the previous work has been focused on defining more or less a ‘perfect’ method for agile security engineering and underlining all possible obstacles, we adapted a different route for our analysis. Instead of a rigid model, this study concentrated on the aspects that are needed to be changed in order to use a
popular agile method in a development project that needs to fulfill governmental security
development instructions. That is, we were looking for a ‘good enough’ solution and the result
shows that mismatch between agile methods and security engineering might be too harshly
reported. Especially the concerns of agile methods’ compatibility with formal requirements can
be quite readily dismissed based on a theoretical analysis only.

Nevertheless, the use of studied agile methods clearly requires quite heavy process
customization, in order for them to be applicable to projects with formal security requirements.
The key findings were that continuous security planning, in the form of iterative design and
architecture activities, has the most potential to improve the security of the finalized product. On
other hand, at the higher security levels, incorporating every-sprint security reviews make it
difficult to retain the ‘agility’ of a method – and formal auditing requirements worsen this
situation altogether. Incorporating the ‘hardening sprints’, suggested by e.g. Fitzgerald et al.,
(2013), or focusing on security only in the planning phase of the project (Boström et al., 2006;
Ge et al., 2007), may simply lead to superficial fulfillment of the requirements, potentially
leading to security issues afterwards. Methodology-based evaluation suggests that security
assurance is best achieved through investing in both planning and implementation tasks.

Scrum is the only one of the methods that include any kind of role definitions. Security
methodologies, on the other hand, tend to have specific role definitions and push for strict
separation of duties. Table 4 presents a summary of key tasks and properties required from a
security assured software development method. The table states whether the selected methods
have the roles defined (Yes or No), or support the extension of existing roles to cover the more
security-specific one. This comparison reveals a more worrying side of the secure agile methods,
especially regarding role definition.

Table 4. Security task role definition

<table>
<thead>
<tr>
<th>No</th>
<th>Task</th>
<th>XP</th>
<th>Scrum</th>
<th>Kanban</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Security specialist roles defined</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>Documentation and guidelines produced</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Support for development time security reviews</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Support for delivery time security reviews</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>Compliant development process roles defined</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

SDL defines several security roles for the development team, such as reviewer/advisor, auditor,
expert, and team champions. It also promotes strict and vigorous separation of duties, all while
the agile methods typically define only a minimum set of roles or none at all. Scrum, in its basic
form, defines only the roles of Product Owner, Developers and the Scrum Master. Of these, the
Developer is the most appropriate one to assume the responsibilities of a security specialist. This,
however, is a clear violation of the industry standard ‘separation of duties’ rule: the developers
themselves are rarely the best persons to break their own code. This approach is anti-agile in two
ways: teams not sharing information is a clear violation against the agile philosophy, and having
separate teams working in parallel bogs down the development speed while increasing the cost.
The same lack of defined security roles also characterizes XP and Kanban, all while giving organizations more freedom in choosing the development tools, mechanisms and processes.

In addition, the message in the studied literature is clear about certain benefit of employing agile methods to develop security-oriented software: developing the software in numerous iterations towards the finalized product may actually improve security assurance, as the product is kept potentially shippable after every sprint – an agile ideal, although rarely achievable. This greatly helps in tracking the changes in security development and detecting possible security threats. In addition, the promoted use of automated testing and other tools is an inherent part of security development, directly applicable also to fuzz testing.

FUTURE RESEARCH DIRECTIONS

This study opens new fruitful research avenues for future development in security regarding software architecture and design fields. While there are different adaptations defined and presented for secure agile software development, there is a lack of empirical evidence and test regarding these adaptations. Future work should be aimed to develop usable agile software development improvements that comply with the requirements of security engineering. Furthermore, this kind of a development work should incorporate the ‘good enough’ principle in order to keep the benefits achieved with agile methods. If the method aims for too good or too strict security process, it might be that transparency and efficiency of agile methodologies will be lost.

In addition, this chapter calls for a new kind of thinking into the security engineering. While security is and will be a main driver in many, if not in all, software development projects, also other aspects are important. These include, e.g., efficient and fast development. The current development in this field has been based purely on the security engineering perspective and much of the realism in the industry is bypassed. Therefore, especially in the agile security engineering field a new fresh start is needed for defining, e.g., cost-efficient solutions and integrating security engineering practices into other activities. While improving awareness has been showed to have good effect on security engineering, lightweight solutions could also work.

The limitation of this study is a lack of empiric evidence, and the logical next step would be to instantiate the methods and possibly include more of them. While security should be based on ‘defined’ rather than ‘empiric’ logic, practice will show not only the applicability of the methods themselves, but also the real cost of security mechanisms to the development process. Security cost is becoming increasingly necessary to pay, as Finland’s public sector’s software security regulations show. As the cost of development is much smaller than rewriting and refactoring an existing code base, integrating the security processes to the development method is crucial. The ultimate objective should be nothing less than finding a framework for the software developers to choose the correct set of roles, methods and processes for each situation and purpose.

CONCLUSION

This study used established and widely-used Finnish government’s security criteria, VAHTI, as a basis for evaluation of three approaches to software development for a regulated environment. Selected security framework was Microsoft SDL and the methods XP, Scrum, and Kanban. Research objective of this study was to use lightweight DESMET evaluation criteria to analyze the adaptability of agile methods to security development, and to estimate the cost of security-
related tasks. The study shows that in a theoretical framework the agile methodologies are readily adaptable to even the most strict security requirements. This result departs from the extant literature, which too often presents agile software development methods incompatible with security engineering practices. Therefore, this chapter suggests future activities in developing and using agile methods for the use of security work in software architecture and design.

REFERENCES


