Securing Scrum for VAHTI

Kalle Rindell, Sami Hyrynsalmi, Ville Leppänen

University of Turku, Department of Information Technology, Finland,
kakrind@utu.fi, sthyry@utu.fi, ville.leppanen@utu.fi

Abstract. Software security is a combination of security methods, techniques and tools, aiming to promote data confidentiality, integrity, usability, availability and privacy. In order to achieve concrete and measurable levels of software security, several international, national and industry-level regulations have been established. Finnish governmental security standard collection, VAHTI, is one of the most extensive example of these standards. This paper presents a selection of methods, tools, techniques and modifications to Scrum software development method to achieve the levels of security compliant with VAHTI instructions for software development. These comprise of security-specific modifications and additions to Scrum roles, modifications to sprints, and inclusion of special hardening sprints and spikes to implement the security items in the product backlog. Security requirements are transformed to security stories, abuse cases and other security-related tasks. Definition of done regarding the VAHTI requirements on is established and the steps to achieve it are described.

Keywords: Scrum, agile, VAHTI, software security, security standards

1 Introduction

Software industry has always been under scrutiny by regulators, standardization organizations and a plethora of industry-specific and more or less general standards. The purpose of all this has been to guarantee a certain level of evidence about certain quality aspects or functionality of the software. Consequently, also software security is becoming increasingly regulated. Several security standards and audit criteria have been established, and even more academic and commercial software development methods have been suggested to meet the standards. A number of the development methods and best practices have achieved a standardized status themselves. Main issue with the standardized methods is that while they provide the necessary *security assurance* required for certain regulated environments and security audits, they in general do not adhere to the agile values or ideology.

The term 'agile' reflects approaching the development of software using new focus, ideology and values [3]. As opposed to the sequential waterfall model, which attempts to maximize the efficiency by not allowing any change, and by locking down the output of each development phase, the agile mindset anticipates change, and even welcomes it. At the core of agile software development lies Scrum, one of the oldest, and the most established and widely-used development approaches among agile methods [19].

The scopes of security standards can be divided into industry specific, national and international regulations. Standardization types include standards for software *safety* and

privacy, control and management of *data*, *software*, *assets*, *personnel* and *processes*, and any other aspects of information security. Consequently, these regulations also concern the design and development of these aspects. The main subject of this study, software development security, can further be divided into its components: security of the used technologies, security of used processes and methods, and, consequently, the security of the produced software. To control the security of its information systems, the Finnish government has published its own public security requirements, VAHTI instructions [10]. VAHTI is the *de facto* standard for the governmental information systems' software maintenance, use and development. VAHTI instructions for application development, published in 2013, include definite requirements regarding the development of the software and inherently the used development method itself [9].

VAHTI, however, is not directly compatible with the mainstream agile software development methods. Therefore, this constructive paper presents modifications to Scrum that are needed to complete with the governmental requirements. We have identified the VAHTI requirements regarding the development method and the development phase, and outline the necessary mechanisms and measures to be instantiated and integrated into Scrum necessary to meet these requirements. These means include security-related roles, processes and techniques, done at the three security levels defined in VAHTI: basic, heightened and high. The security modifications to Scrum are selected by selecting applicable security measures from international standards and established security frameworks and methodologies. While Scrum does not define security roles or processes, we analyse the VAHTI requirements and translate these to concrete software development concepts to be applied to Scrum.

This article consists of following sections: in Section 2, we discuss the background and motivation of this article, as well as cover the related work done in the field of introducing software security mechanisms in agile methods. In Section 3, we present the Scrum method, its key terminology and how a software project is conducted using Scrum. Section 4 introduces the VAHTI instructions for Application Development, the process and reasoning in selecting the key requirements from VAHTI, which are then grouped by their security level and assigned to Scrum roles. In Section 5, we provide the modified Scrum process to meet the VAHTI requirements. The security compliance and security assurance is achieved by importing specific elements of established security frameworks into Scrum. An example project structure is also provided. Finally, in Section 6, we conclude the results and discuss avenues for future research.

2 Motivation, Research Questions and Related Work

Security in its various aspects has been a key topic in information technology since the first computers. Perhaps it is the military background of the development of computing devices, networks and the Internet which still molds the security procedures, aiming to be well-structured, thoroughly documented and strictly regulated — even to the point of being rigid and a hindrance to production. On the industry side, finance and banking were among the earliest to introduce computers to their business processes, and financial data is among the most conspicuously protected information assets in any organization. With the emergence of personal computing and near-ubiquitous Internet services, privacy

and identity protection issues have gained importance among security topics. On the other end, security also concerns countries and governmental entities. These typically have a special focus on the *continuity* of services, especially ones critical to running the society. When comparing the compliance requirement the various security standards and regulations are setting to the ideology presented in the agile manifesto, the task of providing security assurance and complying with standards appears to be a non-agile, or even an anti-agile task. Even the VAHTI instructions for application development state that the 'mandatory' and 'well-documented' S-SDLC (Secure Software Development Life-Cycle) should *'define exit criteria for different development phases'*, a clear presumption that the waterfall model will be used. Against this background the key questions in our research were:

- How to make an S-SDLC conforming with the agile mindset as much as possible?
- Selecting Scrum, the most widely-used software development method, as the reference method; how should it be modified to comply with VAHTI?
- If a requirement stated in VAHTI is considered an item in the backlog, what is the definition of done for each of these items?
- How can the definition of done be achieved with the least security overhead?

To address these questions, we used a Conceptual-analytical research approach and a constructive research strategy [13]. That is, in this study we analysed the integral elements of two existing tools (i.e., Scrum and VAHTI). Based on the results of this conceptual analysis, we present the needed modifications to Scrum that it will fulfill the requirements set by VAHTI. This study presents only the result of conceptual analysis and further works are needed to verify and validate the proposed model.

In extant literature, several secure software development methods have been introduced. Some of these even claim to be agile, or at least use an agile method such as Scrum or eXtreme Programming (XP) as a starting point. Fitzgerald et al. [8] have applied Microsoft Security Development Lifecycle (SDL) [14] to Scrum and created their own version of the methodology to meet organization-specific security requirements. In recent work done at the University of Oulu, Vähä-Sipilä, Ylimannela and Helenius (in [15]) have gathered various additions and modifications to agile programming methodologies, such as an initial 'sprint zero' for security definition purposes, use of security stories for communicating requirements, abuse cases for testing, hardening sprints and overall modifications to Scrum's basic structure, identifying control points crucial for security activities. Additionally, to handle the measurably increased complexity the security requirements add to the software and the processes, even a change of paradigm to aspect-oriented programming (AOP) has been suggested [5]. Typically the suggested S-SDLC methods were instantiated only for a single project within a single company, if instantiated at all.

Implementation of the security controls and tasks is a more clear-cut field: Microsoft suggests its SDL, and even in its current published version 5.2 provides 'SDL for Agile' extension to their method. SDL may be considered a security framework offering a full set of tools, methods and techniques to implement the security tasks they suggest. Although not directly security-oriented, the Capability Maturity Model Integration for Development (CMMI-DEV) was studied by e.g. Diaz et al. [7] regarding the relationship

of heavy software processes of CMMI's managed level 2 processes [2] to Scrum. Similarly, our approach was to examine industry standards and best practices, such as SSE-CMM [12], BSIMM-V [4] and Microsoft SDL, and modify Scrum with a minimum set of items that are necessary for VAHTI compliance. In our approach, the processes are kept as lean as possible by not striving for formal 'maturity levels', and by minimizing any security overhead deemed unnecessary.

3 Scrum

Scrum is an iterative and incremental project management framework, based on the principle that customer requirements and other agents may change during the project's execution, and by working in iterations, allows the team to react to the change. Scrum gives the organization quite a lot of freedom in how to organize and execute the project, but certain key roles and concepts are defined. This section's introduction to Scrum is based on the Scrum Primer book [6].

Scrum is designed promote productivity and mitigate management and governance overhead, by e.g. giving the developers as much freedom as possible in defining and implementing their tasks (self-organizing teams) and all but eliminating the role of the traditional project manager. The project may still require some project and product management functions such as financial or other reporting, but the Scrum project does not have or need a dedicated project manager. Scrum has an emphasis on intra-team communication, favoring team co-location or at least tight online collaboration.

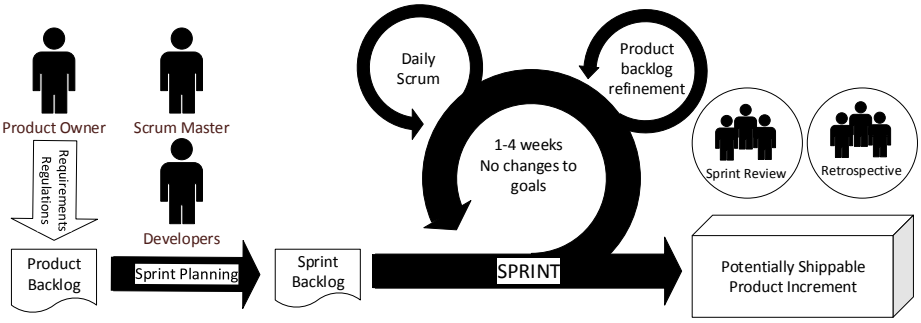

Fig. 1. The Scrum process (adapted from [6])

Figure 1 shows an overview of the Scrum process. This is a slightly modified version of 'pure' Scrum, allowing redefining the product backlog during the sprints. The key concepts of the methods are:

1. The *Team* consists of three core roles: *Product Owner* (PO), representing the customer and stakeholders, *Scrum Master*, facilitating for the team and removing any impediments, and (typically) 3-9 *Developers*, who as a cross-functional and self-organizing team utilize their skills to create *Potentially Shippable Product Increments*.

2. *Stories* are product requirements often written from the product owner's perspective.
3. *Product Backlog* is the list of stories, use cases, requirements and other items that require completion in order to deliver the product.
4. *Tasks* and *Sub-tasks* are concrete steps which the team members create and complete based on the backlog items.
5. In *Sprint Planning*, the team selects from the product backlog items that can (and should) be completed within the next sprint. The prioritisation of the items comes ultimately from the stakeholders, i.e., the customer. In Sprint Planning, the items are converted into completable tasks and sub-tasks, which added to the *Sprint Backlog*.
6. *Sprints* are the iterations in which the team completes items (i.e., tasks) in the sprint backlog within a pre-scheduled time box (typically 7-28 days). During sprints the product backlog items may be refined, added or deleted, while the sprint backlog remains as unchanged as feasible.
7. In *Daily Scrum* meetings the team members brief each other in what they did yesterday to complete the product, what they plan to do today, and whether there are any impediments to the work.
8. The *Definition of Done* is a set of consistent criteria to determine when an item in the product backlog is considered 'ready', typically after regression testing. Determined by the Scrum master with input from the stakeholders through the product owner.
9. *Sprint Review* is held at the end of each sprint. In this event, the team reviews the completed work, and also the incomplete items on the sprint backlog.
10. *Sprint Retrospective* is for the team to review the sprint itself and the work done in it. It aims for continual improvement of the process, and allows for reflection on what was done well, and what needs improvement.

As any software project, a Scrum project starts with pre-planning and requirement gathering. The requirements are either functional or non-functional, and may come in as user stories, regulations, or other requested features that must be implemented in the product. These items are added into the product backlog. One key point in Scrum is to 'deliver value to business' as effectively as possible. In order to do so, the product owner, as customer's voice, prioritizes the items that are selected into each sprint's backlog at sprint planning event. This way it is at least theoretically possible, that after a few sprints (or even one) the 'potentially shippable product increment' can already be utilized despite some less important features being still under development. This way the value can be realized earlier and, as a bonus, user feedback and bug reports can be utilized to make the end product better. In each sprint, the developers pull items into their own work flow from the sprint backlog, entitling the term 'self-organizing team'.

In addition to sprints, research and prototype work may be done in 'spikes'. Similar to sprints, spikes are time-boxed efforts to produce something that contributes towards a completion of a complex backlog item or work as a proof of concept, without necessarily aiming to complete the item and delivering shippable features.

During sprints, environmental and requirement changes can be taken flexibly into account, in which case the product backlog is adjusted accordingly – typically the sprint backlog remains unchanged. After each sprint the results of the work (the potentially shippable product increment) is evaluated, the definition of done for the product backlog items is verified. The definition of done is an important concept in Scrum, as it is the only way items can be removed from the backlog.

For measuring the progress of the project, agile methods utilize a number of techniques. Probably the single most important technique is *story points*: product owner prioritizes some items in the product backlog over others due to their business value, and the developers evaluate them by the estimated development effort. The story points are not directly translatable to work hours. To emphasize this, a non-linear scale such as Fibonacci-like sequence is used. Sometimes the work estimates include an option to declare items too complex to implement in the current sprint. Thus the story points represent the team's view of the required effort, subjective to their current skill set.

Scrum is an *empiric* method: it readily admits and submits to the fact that not everything is known or understood, and that things will change. In security engineering, *defined* methods would be preferred: a process is started and allowed to complete, producing invariably the same results each time [20]. The waterfall model aims to this goal, but by being excessively rigid, it introduces a very concrete risk of failure to meet the stakeholder's and environment's changing requirements. Traditional old school security engineers tend to scold the agile methods' iterative approach as 'trial-and-error'. Despite this lack of acceptance, Scrum is far from unstructured: it just gives the team more freedom in deciding the order in which the items are implemented, and possibility to iterate on the initial requirements, much based on assumptions. At certain point the definition of done criteria will be met, security compliance achieved and security assurance provided. In the process of doing so, the quality of the produced software may actually be higher than those made using sequential methods [17].

4 VAHTI instructions for application development

VAHTI instructions is a wide collection of governmental security regulations, published by Finnish Governmental Steering Group for Information Security (Valtionhallinnon tietoturvallisuuden johtoryhmä, VAHTI). First publications in the VAHTI collection are dated 2001, and they exist to support the data security legislation and Finland's strategies for National Knowledge or Information Society[1] and Information Security[2]. The instructions for application development were published in 2013. Compliance with VAHTI has been mandatory for state agencies, partners and suppliers since 2014.

VAHTI requirements for software development [9] consist of 120 individual requirements, divided into 15 categories. The categories span the whole life cycle of the application or information system, ranging from strategy and resourcing to the eventual end of life and ramp-down of the system. Of this requirement set, only the ones most relevant to the development process were selected for this study. The candidates for inclusion were directly involved with either the tasks during development process, such as security design, audits or reviews; candidates for exclusion considered organizational issues, strategies, policies, method-independent techniques, IT environment, or issues related to the post-development phases of the application's life cycle such as continuity management or system ramp-down. The result set comprises of 23 requirements considered to affect the development method directly. Table 1 shows the final selected VAHTI

[1] http://www.tietoyhteiskuntaohjelma.fi/esittely/en_GB/introduction/index.html

[2] http://www.lvm.fi/c/document_library/get_file?folderId=339549&name=DLFE-10210.pdf&title=Julkaisuja%2051-2009

requirement grouped by the security level. For each requirement, the expected frequency of the task is projected, and the role(s) responsible or affected by the task are displayed. The only requirement directly concerning the development method, SKM-001, states that the development process itself is required to be a Secure Software Development Life Cycle process. The method is to be documented, development personnel trained in its use, utilized at all times, and comply with all the security requirements.

Table 1. VAHTI requirements for development method per security level

Code	Requirement name	Frequency	Dev	SM	PO
OSK-001	Security Training	1	x	x	
OSK-008	Additional security training after change	0 or 1	x	x	
VTM-005	Application Risk Analysis	1 or more	x	x	x
TST-002	Test Plan Review	1	x		
VTM-008	Threat Modeling - recommended	1	x	x	
VTM-010	Threat Modeling updates - recommended	1 or more	x	x	
ESI-001	Goal and Criticality Definition	1	x	x	x
ESI-002	Business Impact Analysis	1			x
VTM-001	Documentation of Security Solutions	1	x		
VTM-006	App. Security Requirement Definition	1 or more	x	x	x
TST-007	Security Auditing	1	x	x	x
TST-009	Security Testing - recommended	Every-sprint	x		
KTY-002	Application Security Settings Definition	1	x	x	
TSK-001	Architecture and Development Guidelines	1	x	x	
SNT-004	External Interfaces Review	1 or more	x	x	x
SNT-006	Attack Surface Recognition and Reduction	Every-sprint	x	x	x
VTM-009	Architectural Security Requirements	1	x	x	
SNT-016	Internal Communication Security - if applicable	Every-sprint	x	x	
TST-001	Security Test Cases Review	1 or more	x	x	x
TST-004	Test Phase Code Review	1	x		
TST-006	Use of Automated Testing Tools	Every-sprint	x		
TST-008	Security Mechanism Review	1	x		
TST-010	Development-time Auditing	1	x	x	x

Dev = Developers, SM = Scrum Master, PO = Product Owner

The included compliance requirements were selected based on their effect to development-time activities, and grouped into following categories:

1. Prerequisites
2. Documentation
3. Code, interface and test case reviews
4. Development-time and product audits
5. Security testing

VAHTI instructions define three security levels: basic, heightened, and high, each with cumulative security requirements. In a Scrum project, the product owner will specify the target level, preferably using tools specified in VAHTI instructions 3/2012

'Instructions for determining the security level of the technical ICT environment' [1], the state office's own instructions, and any applicable legislation. In the next sections, we go analyze each requirement, suggest a means to comply with it and then present the whole VAHTI-compliant Scrum structure for all defined security levels. The security levels are separated by horizontal lines: the first section covers the requirements for basic level, and the heightened and high levels are below that, respectively.

Generic technical requirements, such as 'the application design must follow secure design patterns', and architectural requirements, were considered external to the development method. Also, while technology-specific training for the project personnel is included, any organizational security awareness-type training and general risk management activities were also considered to be out of a single development project. Also, technical or programming technique specific requirements were excluded. These are considered to be part of design patterns and individual developer's proficiency and ability to recognize and utilize them. Moreover, they are depending on the characteristics of the software under implementation. As an exception to the principle of doing only least amount of work, the optional Threat Modeling tasks (VTM-008 and 010) were included already to basic level - although they remain optional. For some reason, VAHTI does not require threat modeling at all. At highest security level, this can be covered by Attack surface recognition and reduction task (SNT-006), or even included in creation of secure patterns and architecture (TSK-001, VTM-009). Similarly, a clearly implementation-dependent requirement for Internal Communication Security (SNT-016) was included, applicable in case of n-tier applications. It is conceivable that a vast majority of software developed for the VAHTI high-security tier is deployed using a secure multi-tier architecture (i.e., separate database, application, and web server components), so the inclusion of this requirement was deemed necessary.

Majority of the tasks fall on the developers; it is conceivable that establishing a role of dedicated security developer would benefit the team, allowing the developers to concentrate on their main vocation. Scrum master, as the 'servant-manager', is also involved in most tasks, albeit indirectly by facilitating the team's work. The Scrum Master is also directly involved in all modeling and planning tasks, reviews, and audits. Following the agile philosophy, the product owner is engaged in development wherever deemed beneficial: as customer's and stakeholders' representative, they have the best knowledge of the software's purpose, use and impact. In addition to the Business Impact Analysis, which is direct input from the stakeholders via the product owner, they participate in external interface and security testing definition tasks, as well as all the audits. These are, after all, done to the customer's benefit. In the next section, we execute these tasks using Scrum.

5 VAHTI-Scrum

Scrum, as a flexible and relatively adaptable framework, defines only the very rudimentary structure for software development. To achieve compliance with VAHTI, additions and modifications are required to the requirement gathering, sprint planning and the sprint structure themselves. Some security activities justify having a dedicated security or 'hardening' sprints. These may prove especially useful just before auditions and

planned releases, if the organization has decided to use those. It is also recommendable to arrange a security sprint, 'sprint zero', before commencing the actual implementation; alternatively, some security work may be completed as security spikes. Spikes could occur at any time - mainly in the early phases of the project, so they do not have even approximate placement on the project time line. In similar fashion, the hardening sprints may be inserted between the implementation sprints, to prepare for audits or just enhance security. Especially on heightened and high levels the amount of security work can be so big that a separate sprint is justified. It is important, though, that security elements become integral part of all the team's work: all tasks should include some security considerations, at least in the form of risk analysis, just as all sprints contain security tasks.

Figure 2 presents the modified Scrum process with the additional requirement sources, roles, security related actions during the sprints and the artifact types produced by the security processes.

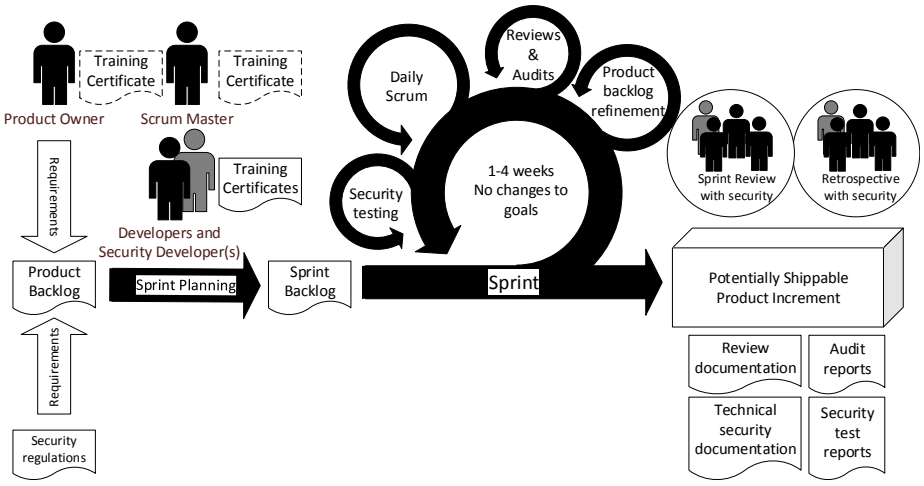

Fig. 2. VAHTI compliant Scrum process

The modifications to standard Scrum are listed below:

– The *security regulation*, although working for the benefit of the stakeholders, is considered a separate source of security requirements. The items, or tasks, added by to the backlog by the regulation are:

  1. Creating security-related documentation artefacts: application risk analysis, threat models, security architecture, goal and criticality definition, business impact analysis, security solutions documentation, application security requirements definition, security settings definition, and architecture and development guidelines.

2. Security training regarding the selected platforms and technologies, such as operating systems, database engines, programming languages and frameworks.
3. Code, test case and interface reviews
4. Security testing
5. Development time and security audits

– There should be at least one dedicated *security developer* in the team, preferably the same person nominated to be responsible for the organization's software security. This results in separation of duties, which enhances security by reducing the amount of *group think*: also somebody else than the developers themselves should design security tests, risk analyses, threat models and attack surface analysis.
– On heightened and high security levels, all sprints contain certain amount of specifically security-related work and security testing.
– Audits are performed at a predetermined point in the project. These are set after completing the corresponding items in the product backlog. The development-time audits on the high security level are suggested to be held at control points, which in agile projects map to product backlog items.
– The team/organization must have nominated a person responsible for the security.

This the following subsections provide a breakdown of security roles, tasks and artefacts on each of the VAHTI security levels.

5.1 Roles

The roles in security-centric Scrum have certain modifications to the standard, plus a new one: the security developer. In VAHTI-Scrum, the team has at least one person in the role of security developer. While still a developer in the Scrum terminology, this role is responsible for security reviews, security test cases and such. It may be beneficial to have more than just the one security developer in a project. The Scrum Master will need a substantial amount of security knowledge and practice, or at least they need to be versatile in adapting to the changing security requirements. Security is an on-or-off deal: there is no middle ground in testing or auditions, and ignoring security problems cannot be considered a sustainable idea.

Similarly to the other roles, also the Product Owner has new responsibilities. As the stakeholder's representative, they need to be aware of the security regulations, legislature, customs and other rules, in addition to the normal duties. Being a product owner in a Scrum project may very well become a full-day job on the higher levels.

5.2 Tasks

For each task, we identify the role(s) responsible for its execution (see also Table 1), and the artefacts produced by the task.

Basic level

Security Training (OSK-001).
- The Scrum Master facilitates for (preferably certified) internal or external training and participates when necessary. This is likely to be performed as a spike as it does not directly contribute to the product increment.
Additional security training after change (OSK-008).
- The Scrum Master facilitates for internal or external training after the need has been identified, and participates when necessary.
Application Risk Analysis (VTM-005).
- The team identifies the security concerns, and technology and environment specific risks. Sources such as OWASP Top 10[3], Tsipenyuk et al. [18] or Howard et al. [11] in addition to VAHTI's own recommendations should be used to identify software risks. The result of the analysis is used as input for threat modeling. Input mainly from the developers, producing a risk analysis document.
Test Plan Review (TST-002)
- The person nominated to be responsible for the security reviews the test plan. The produced review report is part of the security evidence to prove the software and process is VAHTI compliant.
Threat Modeling (VTM-008)
- Although optional, it is recommended that a formal threat model is created based on the application risk analysis. Done by the developers, results a threat model document.
Threat Modeling updates (VTM-010)
- Optional. When the requirements or environment changes and a new risk analysis is performed, the threat model should be updated.

Heightened level

Goal and Criticality Definition (ESI-001)
- In practice the same requirement as Business Impact Analysis, although from the VAHTI perspective and done by the whole team: the impact and business use of the software is analyzed and its data confidentiality assessed. The resulting document is used as input for security requirement definition.
Business Impact Analysis (ESI-002)
- Similar as above, but concentrates on the software's criticality and impact on the customer's business. Analysis is provided mainly via the product owner.
Documentation of Security Solutions (VTM-001)
- Component level security documentation produced by the developers.
Application Security Requirement Definition (VTM-006)
- Security requirement uses the goal and criticality definition and risk analyses (optionally, also threat models) as an input to define the security requirements. This is a formalization of work that has already been done: the software is already determined to belong to the heightened security level. Done by the whole team, involving also PO.

[3] http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf

The resulting document is used as input for the maintenance phase.
Security Auditing (TST-007)
- Performed by an external and independent auditor, facilitated by the Scrum master. Mandatory parts include automated penetration testing; administrative audit to verify the software's maintenance processes (post development phase - does not concern development); architectural audit from security point of view.
Security Testing - recommended (TST-009)
- While a dedicated set of security tests is not mandatory, it is our recommendation to perform such tests to help passing the security audit. Security test case review is mandatory on the High security level. Inconsistently, actual security testing remains non-mandatory in VAHTI!
Application Security Settings Definition (KTY-002)
- At the end of implementation and before deployment the software settings are documented and a maintenance guide with hardening instructions is written.

High level

Architecture and Development Guidelines (TSK-001)
- This is an organizational document that defines also some coding practices, such as exception handling. If this document does not exist, it will have to be created before security audit. Input from the organization's developers and Scrum masters (if several) can be done in e.g. Scrum of Scrums, a process of synchronizing the Scrums.
External Interfaces Review (SNT-004)
- The external interfaces of the software are reviewed against the architectural guidelines. Input from the developers, review is facilitated by the Scrum master.
Attack Surface Recognition and Reduction (SNT-006)
- All attack vectors are to be identified and security mechanisms planned accordingly. In practice just different approach to threat modeling; formalization of the work done at architectural planning by the developers.
Architectural Security Requirements (VTM-009)
- Based on the attack surface recognition (and threat modeling), the architecture is analyzed against recognized attack vectors by the developers.
Internal Communication Security – if applicable (SNT-016)
- The developers should be aware of the technical environment of the software for this one. For example, if a web application uses separate database or application servers, the communication interfaces must be hardened.
Security Test Cases Review (TST-001)
- Review the quality of the security test cases. The cases should be derived from other documentation and their validity and comprehensiveness is verified (i.e., sanity-checked) . Main responsible is with the security developer.
Test Phase Code Review (TST-004)
- Dubbed informal and performed by the person nominated to be responsible for security. Review findings are documented. Developer task.
Use of Automated Testing Tools (TST-006)
- Partially organizational requirement, as acquiring the tools may cause administrative

overhead. VAHTI specifically mentions e.g. fuzzers and code analyzers. This task is performed by the developers; the Scrum master facilitates acquiring the tools.
Security Mechanism Review (TST-008)
- Code level review of security mechanisms. Check list is to be derived from architectural level documents and other relevant documentation produced during the VAHTI-Scrum development process. Done by the developers and the security developer.
Development time Auditing (TST-010)
- One or more external audits to be performed at different phases of development. VAHTI only states that the software's security is to be audited, so it is to be agreed with the stakeholders how to handle this. A good baseline would be architecture, interfaces, security mechanisms and/or the review documentation. Involves all roles

5.3 Artifacts

Software security assurance is provided by evidence, and in most cases this means documentation: reports, plans, technical documents, memos and other document artifacts considered relevant for security. VAHTI is not an exception to that, and pragmatically speaking, producing the required artifacts *fulfills the VAHTI Definition of Done*. The security documentation reflects a significant part of work. However, the figurative security burn down chart does not reach zero even when the last document has been finalized: security tasks will remain a part of every sprint even after that.

The produced artifacts in a VAHTI-Scrum and their dependencies are outlined in Figure 3.

Figure 3 is a matrix with the security levels on the x-axis and development phases on the y-axis. Arrows indicate input; the document is a result of a process of the same name. It should be noted that if the project is operating on high security level, the Architectural and Development Guidelines document may and should be used as input for other documentation. For clarity, the arrows do not cross the security level barriers from higher level to a lower one. Items with dashed lines are optional, yet recommended. On the bottom row, we have the external documents on higher levels: audition reports, and input for the deployment and maintenance phases of the software's life cycle.

6 Conclusions and future research

Software security and security regulation aim at a defined process, producing a measurable, evidence-backed result. This result is called security assurance or security compliance. In Finland, the state agencies have formalized their security requirements into the form of collection of VAHTI instructions; these instructions also concern application development. Standardized secure software development methods have deep roots in the waterfall era, but 'agile' is not the antithesis for defined processes or structured way of working.

In our earlier work, we were interested about the adaptability of the agile methods to the software development in general [16]: this study further utilized the findings and analysis, and provided an example how and with what mechanisms the Scrum method can be adapted to provide compliance with VAHTI instructions. This answers to our two

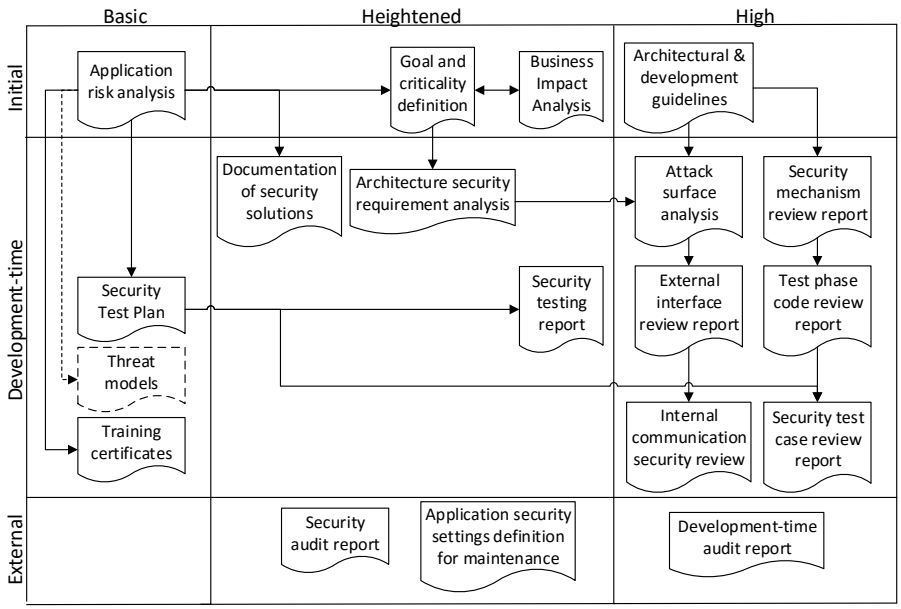

Fig. 3. VAHTI documentation artifacts

first guiding research questions. The Definition of Done in security context is twofold: the formal requirements may be fulfilled, and contribute to the actual DOD; security may be considered 'done' only when the project finishes - or, even when the software life cycle ends.

Naturally, this study has limitations. The presented model is based on a conceptual-analytical approach and it has not been empirically evaluated. While we have carefully addressed the two studied concepts and created the new model based these results, it is possible that some of the proposed modifications can be improved. Thus, further work is needed to validate and verify the model with, e.g., a case study conducted in an university' course with students or in an industrial setting.

The research field offers several complex and interesting opportunities for future study: while we presented a method to fulfill VAHTI's requirement with agile approach, it would be fruitful to understand how industry is currently working with the requirements. A qualitative case study with selected companies is planned to help identify the best practices and methods to use.

Furthermore, benefits and drawbacks introduced to security and safety development by agile methods should be studied. While in a quick glance, it seems that agility in development and security of the product are competing objects that cannot be easily achieved in the same project, an analysis of advantages and disadvantages of using agile in secure software development should be performed. This would, furthermore, bring an answer to the question, what are the reasons to adopt agile in the an environment which is not the best for it?

References

1. Teknisen ympäristön tietoturvataso-ohje, https://www.vahtiohje.fi/web/guest/3/2012-teknisen-ympariston-tietoturvataso-ohje, ref. 18th August 2015
2. Cmmi for development, version 1.3 (2010), http://www.sei.cmu.edu/reports/10tr033.pdf, ref. 18th August 2015
3. Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Merllor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for agile software development (2001)
4. BSIMM: The Building Security In Maturity Model, ref. 2015.08.20
5. De Win, B., Vanhaute, B., De Decker, B.: Security through aspect-oriented programming. In: De Decker, B., Piessens, F., Smits, J., Van Herreweghen, E. (eds.) Advances in Network and Distributed Systems Security, IFIP International Federation for Information Processing, vol. 78, pp. 125–138. Springer US (2002), http://dx.doi.org/10.1007/0-306-46958-8_9
6. Deemer, P., Benefield, G., Larman, C., Vodde, B.: The Scrum Primer: The ligthweight guide to the theory and practice of Scrum. InfoQ, version 2.0 edn. (2012)
7. Diaz, J., Garbajosa, J., Calvo-Manzano, J.: Mapping cmmi level 2 to scrum practices: An experience report. In: O'Connor, R., Baddoo, N., Cuadrago Gallego, J., Rejas Muslera, R., Smolander, K., Messnarz, R. (eds.) Software Process Improvement, Communications in Computer and Information Science, vol. 42, pp. 93–104. Springer Berlin Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-04133-4_8
8. Fitzgerald, B., Stol, K.J., O'Sullivan, R., O'Brien, D.: Scaling agile methods to regulated environments: An industry case study. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 863–872. ICSE '13 (2013)
9. FMoF: Sovelluskehityksen tietoturvaohje (2013), https://www.vahtiohje.fi/web/guest/vahti-1/2013-sovelluskehityksen-tietoturvaohje, ref. 17th March 2015
10. FMoF: Vahti-ohje (2015), http://www.vahtiohje.fi, http://www.vahtiohje.fi, Referenced 17th March 2015
11. Howard, M., LeBlanc, D., Viega, J.: 19 Deadly Sins of Software Security. McGraw-Hill, Inc., New York, NY, USA, 1 edn. (2006)
12. ISO/IEC: Information Technology - Security Techniques - Systems Security Engineering - Capability Maturity Model (SSE-CMM) iso/IEC 21817:2008
13. Järvinen, P.: Research questions guiding selection of an appropriate research method. Series of Publications D – Net Publications D–2004–5, Department of Computer Sciences, University of Tampere, Tampere, Finland (December 2004)
14. Microsoft: Microsoft Security Development Lifecycle. Microsoft (2012)
15. Pietikäinen, P., Röning, J.e.: Handbook of The Secure Agile Software Development Life Cycle. University of Oulu (2014)
16. Rindell, K., Hyrynsalmi, S., Leppänen, V.: A comparison of security assurance support of agile software development methods. In: Proceedings of the 16th International Conference on Computer Systems and Technologies. p. TBA. ACM (2015)
17. Solinski, A., Petersen, K.: Prioritizing agile benefits and limitations in relation to practice usage. Software Quality Journal pp. 1–36 (2014), http://dx.doi.org/10.1007/s11219-014-9253-3
18. Tsipenyuk, K., Chess, B., McGraw, G.: Seven pernicious kingdoms: a taxonomy of software security errors. Security Privacy, IEEE 3(6), 81–84 (Nov 2005)
19. VersionOne: 8th annual state of agile survey (2013), http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf, Referenced 17th August 2015
20. Williams, L., Cockburn, A.: Agile software development: it's about feedback and change. Computer 36(6), 39–43 (June 2003)