A Comparison of Security Assurance Support of Agile Software Development Methods

Kalle Rindell, Sami Hyrynsalmi, and Ville Leppänen

Abstract:

Agile methods increase the speed and reduce the cost of software projects; however, they have been criticized for lack of documentation, traditional quality control, and, most importantly, lack of security assurance - mostly due to their informal and self-organizing approach to software development. This paper clarifies the requirements for security assurance by using an evaluation framework to analyze the compatibility of established agile security development methods: XP, Scrum and Kanban, combined with Microsoft SDL security framework, against Finland's established national security regulation (Vahti). We also analyze the selected methods based on their role definitions, and provide some avenues for future research.

Key words*: secure agile development, security assurance, DESMET, SDL, XP, Scrum, Kanban, Vahti.*

INTRODUCTION

The need for software security has been a main driver in software development since the dawn of first shared mainframes. 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 has been evolving and changing due to extensive use of the Internet and public services, and also the software industry itself has gone through an unprecedented shift from waterfall-type sequential development methods towards iterative agile and lean methods. The need for security has also been realized into the form of several commercial, international and national standards, such as payment card industry's PCI, USA's HIPAA and SOX, and international security-related standards, most importantly ISO/IEC 27002. Also, several security frameworks and security-focused development methods have been developed. For the purposes of this study, we have selected three widely-used development methods, Scrum, XP and Kanban and 'enhance' them with elements from Microsoft SDL, an industry standard security framework. The security requirement used are teh Finnish government's security guidelines, called Vahti[1]. The compliance of the selected development methods and the SDL framework is then evaluated against the requirements of Vahti, and the applicability and adaptability of the 'security-enhanced' methods themselves are evaluated using Kitchenham's DESMET evaluation framework [15].

Agile methods promote iterative development and informal interaction, and give a lesser 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 [7, 16, 17]. Introducing strict security requirements to the software development process usually results in creation of a formal security architecture out of necessity to fulfill the strict external security criteria. When the security process elements, such as reviews, security testing, processes and documentation, are incorporated into the agile method itself, this has

[1] https://www.vahtiohje.fi/web/guest/home

the potential of causing considerable delays in the form of rework, and increase the cost of the development effort [7]. All this is extra 'management overhead', and appears to be in direct contradiction with agile methods' core philosophy of leanness and informality [6]. Applying the security process to the agile or lean development methods has the potential of rendering the methods, by definition, something that is neither agile nor lean.

The research objective of this study is to show that the selected agile methods are adaptable to security development using SDL, and selecting an agile method can have a positive effect on the security assurance, while reducing development time and cost.

RELATED WORK

One of the starting points for this study was a formal categorization of security aspects for software development, and conduction of a DESMET feature analysis [15]. A similar approach has earlier been selected by Beznosov and Kruchten [7], albeit they do not use an established security criteria or framework, nor an external evaluation criteria in their study. Also, there exists a number of studies concerning secure software development concept in general, also covering the topic of security-focused testing [9]. Abrahamsson et al. [3] 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.

A bulk of papers seems to have an approach of documenting a proprietary corporate software development method, or even specify their own, such as Baca and Carlsson [5] and Boström et al [8]. Their study also compares existing software methodologies, including SDL, with a proprietary method, and claims also a new proprietary method of their own. Other studies, such as Alnatheer et al. [4] conduct empiric research on the impact of agile methods on software security. Fitzgerald et al. [10] have completed a case study, adapting Scrum-based methods in an environment with strict formal security regulations. Their study discusses security regulations and Scrum in considerable depth, yet only within the scope of a single company and a single development method. This paper aims to extrapolate on those results by using a nationwide criteria and new development methods, and the SDL security framework. In contrast to most of the previous approaches, security is in this paper 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 [7]. We also concentrate on the challenges this brings into the development process and the quality assurance closely associated with security controls, as discussed by Fitzgerald et al. [10]. The effect of the introduction of security element is evaluated by applying DESMET feature analysis. 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 against 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.

BACKGROUND

Software security cannot simply be declared by the software publisher: the security claims also need proof, backed by evidence. To verify the security claims stated by the software, the security evidence is gathered through such activities reviewing the software, its documentation and processes, 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 [1]. Preferably this is done in the most efficient way possible, taking into account the organization and the operational environment.

Vahti guidelines

The specific set of security requirements used in this paper is Vahti (literally translated 'Guard', a mnemonic acronym in Finnish for 'Government Information Security'). Vahti is one of the earliest and most comprehensive sets of open and public information system security regulations. The guideline comprises of 51 documents, published since 2001, and covers various aspects of information systems' use, development and other stages of the life cycle. The guideline covers also various aspects of information systems management, governance, use, and, ultimately, help implementing Finland's national information security strategy, published in 2009. The guidelines were originally targeted only for government's internal information systems work; due to public sector's integral part in the Finnish society, Vahti's basic security level in in the process of becoming a de facto security standard in all information systems that are connected to or interact with a government system. 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 [14], and derived from Systems Security Engineering – Capability Maturity Model [13]. These requirements aim to cover the life cycle and various use cases of the public information systems, and span over several dozen documents available at the appropriate ministry's website (see footnote). The Vahti guidelines specify three security levels: Basic, Heightened and High. Compliance with the Basic level is required of all government's software.

Software development methods and evaluation framework

For this study, we have chosen three agile methods: eXtreme Programming (XP), Scrum, and Kanban. Several derivatives of these methods exist, and there is an abundance of software development models presented to public both by industry professionals as well as academic researchers. Some examples of these are covered in the section Related Work.

Microsoft's effort to improve the security of their software has them to develop a quite fundamental security framework, dubbed the Security Development Lifecycle (SDL) process. Currently in version 5.2, the SDL is based on iterative spiral model borrowed from and adaptable to agile methodologies [2, 5]. SDL's approach is heavy on processes, documentation and organization, which is why this study aims to identify the minimal set of SDL elements required to fulfill the security requirements. SDL divides agile activities into three categories: one-time requirements, bucket requirements and every-sprint requirement. The naming seems to suggest that SDL for Agile is meant for Scrum or Scrum-like methods.

Extreme Programming (XP) is one of the first and widely used agile development models. The XP method itself is merely a collection of practices and values [6]. The guidelines given by the method are quite practical, such as the use of pair programming or continuous integration. Popularity of XP method especially in the beginning of the first decade of the 21st century has spun attempts to bring security elements into the method. Earlier examples of security enhancements to XP consist of security-related user stories and abuser stories in the planning phase [8, 12].

Scrum may be considered as the current mainstay of the software industry. In the literature, earlier examples of security enhancement to Scrum consist of loosely SDL-based security features specifically aimed for regulated environments, such as Fitzgerald et al. [10]. These features and processes include 'hardening sprints', which consist entirely of security-related planning and documentation, and additional security and quality assurance reviews and checks. This methodology includes new roles that are not included in baseline Scrum. The organizational requirements are also discussed in this paper. Scrum was selected due to its overwhelming popularity in the current software development industry [19].

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 workflow: the workflow 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 [18] and other hybrid methods.

DESMET framework was developed 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 are analyzed as a simulated case study, based on expert opinions and without instantiation.

EVALUATION AND RESULTS

Vahti security criteria for application development comprises the whole life cycle of software. From the complete list of 118 requirements [11], 22 requirements were deemed to apply directly to the development methods and were included in this study (see Table 1). The selection was made based on the relevance of the requirement to the software development method: the selected criteria are either requirements for the documentation, reviews or the development process itself. The only selected organizational requirement, the one for security training, was included due to the fact that SDL emphasizes this security-enhancing mechanism, and it affects all the development roles.

Table 1 lists the methodologies' compliance with each requirement. The following values are used: I = *Integral to the method*; A = *Adaptable; method can be adjusted to support the requirement*; and N = *Needs improvement; incompatible.*Each criteria is assessed by its security level: basic (1), heightened (2) and high (3). While these levels strictly apply to a system's criticality (a combination of required availability and the sensitivity of data), it has been selected to define the security requirement of the software. The other assessment criteria is the frequency requirement for the appropriate technique set by SDL. The frequency is encoded into three values: one-time requirements (1), bucket requirements (2) and every-sprint requirements (3). Each task is further ranked to either Automated (A=0.5), Semi-automated (S=1) or Manual (M=2). Semi-automated means the bulk of the work is done by automated tools, which in turn may require a considerable amount of manual configuration. Cost is calculated by multiplying the level of automation with the frequency of the task

1. *Application Risk Analysis* is an essential security element, and well integrable (I) into all methods. 2. *Test Plan Review* is an internal security personnel review of the test plan. Supported by all (I). 3. *Threat Modeling* consists of compiling the list of the threats and keeping that up to date during every sprint. Cornerstone of SDL and essential to any security related development. Provides base for risk analysis and guides architectural choices, among other things. The threat 'landscape' is dependent on the software's intended users and use environment. Essential to all methods (I). For some reason, this requirement was not mandatory even at Vahti's highest level - a clear omission unless included in other tasks.

4. *Goal and criticality* requirement means classification of the software and documentation of its purpose. Both XP and Scrum (A) were found lacking in this respect, with the Scrum-based method more readily adaptable to produce planning phase documentation. Kanban-based methods are also deemed adaptable (A). 5. *Business Impact Analysis* is basically method independent requirement, and as such, deemed adaptable (A) 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.

Table 1: Methodologies compliance with security requirements

No.	Requirement	Lvl.	Aut.	Freq.	Cost	XP	Scr.	Kan.
1	Application Risk Analysis	1	M	3	6	I	I	I
2	Test Plan Review	1	M	2	4	A	A	A
3	Threat Modeling	1	S	3	3	I	I	I
4	Goal and Criticality Definition	2	M	1	2	A	A	A
5	Business Impact Analysis	2	M	1	2	A	A	A
6	Documentation of Security Solutions	2	M	1	2	A	A	A
7	App. Security Requirement Definition	2	M	2	4	I	I	I
8	App. Security Settings Definition	2	M	2	4	A	A	A
9	Security Testing	2	A	3	1.5	I	I	I
10	Security Auditing	2	M	1	2	A	A	A
11	Arch. and App. Devel. Guidelines	3	M	1	2	A	A	A
12	External Interfaces Review	3	M	2	4	I	I	I
13	Use of Secure Design Patterns	3	M	1	2	I	I	I
14	Attack Surface Reduction	3	M	3	6	I	I	I
15	Architectural Security Requirements	3	M	1	2	I	I	I
16	Internal Communication Security	3	S	3	3	I	I	I
17	Security Test Cases Review	3	M	3	6	A	A	A
18	Test Phase Code Review	3	M	3	6	I	I	I
19	Use of Automated Testing Tools	3	A	3	1.5	I	I	I
20	Security Mechanism Review	3	M	1	2	N	A	A
21	Development-time Auditing	3	M	1	2	N	N	N
22	Security training for Developers	3	M	3	6	A	A	A

M = Manual, S = Semi-Automated, A = Automated; I = Integral, A = Adaptable, N = Incompatible

6. *Documentation of Security Solutions* is a direct requirement to communicate the security requirements to the developers through documentation. Both agile methods are fundamentally against this approach, and will need improvement (N). 7. *Application Security Requirements* is a high-level document, 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 (I). 8. *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. Suggested action would be a separate documentation sprint, to be added into the agile methods (A). 9. *Security Testing* states that security testing should be incorporated into the standard testing procedure. Supported by all (I). 10. *Security Auditing* is a requirement for Heightened and High Vahti levels; requires an external auditor. This requirement was included due to its strain on the development process, mainly through architecture auditing. Supported by all methods (I).

11. *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 (A). 12. *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 (I). 13. *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 methods support this requirement (I). 14. *Attack Surface Reduction* means 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 (I). 15. *Architectural Security Requirements* means and analysis of the application's architecture against known or anticipated threats. All methods support this (I). 16. *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 (I). 17. *Security Test Cases Definition* this 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. Adaptable to all methods (A). 18. *Test Phase Code Review* is informally performed by the internal security personnel, and documented either separately or even straight into the source code. Supported by all (I). 19. *Use of Automated Testing Tools* is more or less standard practise for all agile methods. Tools include fuzzers, vulnerability scanners, code analyzers and continuous integration tools. Supported by all (I). 20. *Security Mechanism Review* is a code-level review of how security components are implemented. Basically method independent, but may be difficult to implement in iterative methods as after changes this review has to be done again. XP-based methods will need improvement to support this (N), whereas the Scrum and Scrum-based methods with specific hardening sprints may include this step into those (A). 21. *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 (N). 22. *Security training* means organizing purpose-oriented and role-based training for the personnel responsible for the application development, such as product owner, developers and testers. Adaptable to all methods (A).

The agile methods 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. However, it was deemed unjustifiable to claim that e.g. Scrum does not support security documentation at all; security requirement simply needs to be included in the stories and backlog, and appropriate tasks assigned to the developers. Incorporating security reviews and auditing into the development processes proved to be a bigger issue, as they are more difficult to incorporate into the iterative process. All three methods were found inherently compatible with or adaptable to all planning and implementation phase activities. The key findings were that constant iterative planning has the potential to improve the security of the finalized product. On the higher security levels, incorporating the required the every-sprint security reviews and audits make it difficult to retain the 'agility' of the method. Simply incorporating so-called 'hardening sprints', as suggested in some studies [10], or focusing on security only in the planning phase of the project [8, 12], may simply lead to superficial fulfillment of the requirements and has the potential to lead to security issues afterwards. Methodology-based evaluation suggests, that security assurance is best achieved through both planning and balanced security-related resourcing.

Scrum is the only one of the methods that includes role definitions. Security processes, on the other hand, have specific role definitions and push for strict separation of duties. Table 2 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 (Y or N), 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.

SDL defines several security-related roles to complement the development team, and promotes strict and vigorous separation of duties, 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

Table 2: Security task role definition

No.	Task	XP	Scrum	Kanban
1	Security specialist roles defined	N	Y	N
2	Documentation and guidelines produced	Y	Y	Y
3	Support for development time security reviews	N	Y	N
4	Support for delivery time security reviews	N	Y	N
5	Compliant development process roles defined	N	N	N

Y = Yes, role or task defined; N = No, role or task not defined.

the industry standard 'separation of duties' rule: the developers themselves are rarely the best persons to break their own code. Even if the team is split into developers and security team, they cannot discuss the security openly, as that might lead into group think. This issue is anti-agile in two ways: teams not sharing information is a clear transgression against the agile philosophy, and having separate teams working in parallel bogs down the development speed while ramping up the cost. The lack of defined security roles also characterizes XP and Kanban, while giving the developing organizations even more freedom in choosing the development tools, mechanisms and processes.

SUMMARY AND FUTURE WORK

This study used an 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. The selected security framework was Microsoft SDL and methods XP, and Scrum, and Kanban. The research objective of this study was to use light-weight DESMET evaluation criteria to analyse the adaptability of agile methods to security development, and to estimate the cost of security-related tasks.

The study, although conducted only in a theoretical framework, shows that agile development is readily adaptable to even the most strict security requirements. Also, 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. This greatly helps in tracking the changes in security development and detecting possible security threats. Also, the promoted use of automated testing and other tools is an inherent part of security development, directly applied to e.g. fuzz testing.

The limitation of this study was 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 codebase, 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.

REFERENCES

[1] Practical security stories and security tasks for agile development environments. http://www.safecode.org/publication/SAFECode_Agile_Dev_Security0712.pdf. Referenced 9th March, 2015.

[2] Microsoft security development lifecycle (SDL) process guidance - version 5.2, 2012.

Referenced 17th March 2015.

[3] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen. New directions on agile methods: A comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 244–254, Washington, DC, USA, 2003. IEEE Computer Society.

[4] A. Alnatheer, A. Gravell, and D. Argles. Agile security issues: A research study. In *Proceedings of the 5th International Doctoral Symposium on Empirical Software Engineering (IDoESE)*, 2010.

[5] D. Baca and B. Carlsson. Agile development with security engineering activities. In *Proceedings of the 2011 International Conference on Software and Systems Process*, ICSSP '11, pages 149–158, New York, NY, USA, 2011. ACM.

[6] K. Beck. Embracing change with extreme programming. *IEEE Computer*, 32, 1999.

[7] K. Beznosov and P. Kruchten. Towards agile security assurance. In *NSPW '04 Proceedings of the 2004 workshop on New security paradigms*, pages 47–54, 2004.

[8] G. Boström, J. Wäyrynen, M. Bodén, K. Beznosov, and P. Kruchten. Extending XP practices to support security requirements engineering. In *Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems*, SESS '06, 2006.

[9] B. Fitzgerald and K.-J. Stol. Continuous software engineering and beyond: Trends and challenges. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, RCoSE 2014, pages 1–9, New York, NY, USA, 2014. ACM.

[10] B. Fitzgerald, K.-J. Stol, R. O'Sullivan, and D. O'Brien. Scaling agile methods to regulated environments: An industry case study. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 863–872, 2013.

[11] FMoF. Sovelluskehityksen tietoturvaohje, 2013. Ref. 17th March 2015.

[12] X. Ge, R. Paige, F. Polack, and P. Brooke. Extreme programming security practices. In G. Concas, E. Damiani, M. Scotto, and G. Succi, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 4536 of *Lecture Notes in Computer Science*, pages 226–230. Springer Berlin Heidelberg, 2007.

[13] ISO/IEC. information technology - security techniques - systems security engineering - capability maturity model (SSE-CMM) iso/IEC 21817:2008.

[14] ISO/IEC. Information technology - security techniques - code of practice for information security controls iso/IEC 27002:2013, 2013.

[15] B. Kitchenham, S. Linkman, and D. Law. Desmet: a methodology for evaluating software engineering methods and tools. *Computing & Control Engineering Journal*.

[16] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07. IEEE Computer Society, 2007.

[17] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: A study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 492–501, New York, NY, USA, 2006. ACM.

[18] N. Nikitina, M. Kajko-Mattsson, and M. Stråle. From Scrum to Scrumban: A case study of a process transition. In *Proceedings of the International Conference on Software and System Process*, ICSSP '12, pages 140–149. IEEE Press, 2012.

[19] VersionOne. 8th annual state of agile survey, 2013. http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf, Referenced 17th March 2015.

ABOUT THE AUTHORS

Doctoral Candidate Kalle Rindell, Postdoctoral Researcher Sami Hyrynsalmi, Prof. Ville Leppänen, Dept. of Information Technology, University of Turku, Finland. E-mail: {kakrind, sthyry, villep}@utu.fi.