Busting a Myth: Review of Agile Security Engineering Methods

ABSTRACT

Engineering methods are essential in software development, and form a crucial element in the design and implementation of software security. Security engineering processes and activities have a long and well-standardized history of integration with software development methods. The inception of iterative and incremental software development methods raised suspicions of an inherent incompatibility between the traditional non-agile security processes and the new agile methods. This suspicion still affects the attitude towards agile security. To examine and explore this myth, this study presents a literature review of a selected set of agile secure software development methods. A systematic literature method was used to find the definitive set of secure agile software development methods, of which a core set of 11 papers was selected for analysis, and the security activities documented in the methods were extracted. The results show a wide and well-documented adaptation of security activities in agile software development, with the observed activities covering the whole security development life cycle. Based on the analysis, the inherent insecurity of the agile software development methods can be declared to be just a myth.

KEYWORDS

Agile Software Development, Mythology, Security Engineering, Software Development Methods, Review

ACM Reference format:
. 2017. Busting a Myth: Review of Agile Security Engineering Methods. In *Proceedings of ACM Conference, Reggio Calabria, Italy, SSE Workshop (ARES Conference 2017),* 10 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Importance of software security has been dramatically alleviated by the rise of the public awareness in the current post-Snowden era. New, fresh software development methodologies continue to be presented, with the aim of guaranteeing a credible and acceptable level of security. While the level and complexity of the requirements is increased, the development process is expected to be faster and more reactive in order to guarantee market success [21]. To cope with these somewhat contradicting conditions, a series of software development methods that combine practices from agile software development and secure development have been presented.

The objective of this paper is to organize and map the existing agile software development methods that contain pronounced security engineering activities. Several such methods exist; however, new methods have been continuously presented, based on the argument that agile does not suit well for secure software development. This study contributes to the field of secure software engineering by mapping existing secure agile methods, showing the empirical evidence supporting the methods, and collecting and pointing out the agile security activities in each of them.

The work is structured as follow: Section 2 presents the central concept for this work in area of agile software development. Section 3 summarizes the used research process and it is followed by descriptions of secure agile software development methods, the empiric cases and the security activities used (Section 4). Summarized results of the analysis are presented in Section 5, and the results and findings are discussed in Section 6. Finally, Section 7 concludes the study and proposes avenues for further research.

2 BACKGROUND

A new wave of lightweight software development methods, dubbed 'agile', started to gain popularity in the end of the 1990's. A remarkable milestone in the field was in 2001 when the Agile Manifesto with its twelve principles was published [1]. Since then, the agile methods and practices have gained tremendous popularity both as research subjects in the academia [8] as well as practices in the industry [12].

In contrast to the process-oriented methodologies, agile methods emphasize e.g., short-term planning and adaption to changes over planning ahead and following strict processes. A series of agile methods has been presented [cf. 1] since the Agile Manifesto, with Scrum and its derivatives being currently the most widely adapted [22]. Scrum is a simple process model where the software is produced in short iterations lasting just a few weeks, and consisting of well-defined ceremonies, roles and artifacts [1].

Extreme programming (XP) is one of the earliest agile methods, and still the second most used [22]. In contrast to Scrum, it resembles more a collection of tools and practices rather than a strict process model. The method does not, for example, define roles for the participants, as Scrum does. XP promotes for example practises of simple design, pair programming, continuous integration, test-driven development and collective code ownership [1].

The studies surveyed in this article also reference widely to Microsoft Security Development Lifecycle, OWASP CLASP, Cigatel Touchpoints and ISO Common Criteria. These security processes and models provide the basic security tools and activities, ready to be adapted and integrated into the software development methods. The models have roots in security engineering practices that predate the agile boom, leading to initial difficulties with adaption to agile methods. A starting hypothesis for several works, cf. for example [19], is that an agile method in itself is somehow perceived to be incompatible with secure software development. Despite the difficulties, several examples of agile software development with integrated security engineering activities and processes have been presented [e.g. 3, 19, 23]. However, the number of empirical

ARES Conference 2017, Reggio Calabria, Italy

DOI: 10.1145/nnnnnnn.nnnnnnn

Table 1: Searches

Library	Search term	Result set size
ACM Digital Library	‘ AND software AND security AND engineering	59
IEEE Explore	agile AND software AND security AND engineering	120
Springer Link	agile AND software AND ”security engineering”	101
Wiley	agile AND software AND ”security engineering”	32
ScienceDirect	agile AND software AND ”security engineering”	64
Total		376

studies assessing and describing industrial cases is low. For example, Rindell et al. [17] studied a case where Scrum was applied in the development of a security-critical system. While they found shortages in converting the security engineering processes into a truly agile process, the project itself was deemed a success: the end product was delivered on budget and on time, while meeting the requirements of the security regulations that applied to the case.

Regulations represent, in agile terms, customer requirements for security. They also help defining the level of ‘good enough’ or ‘minimum viable’ security. For these reasons, considerations regarding compatibility with a security standard were taken as one of the aspects for analysis when assessing the material. Also, evidence of practical empiric application of security activities in agile software engineering was noted.

3 RESEARCH PROCESS

The purpose and motivation of this study was to search the extant literature for examples of agile software development with security engineering activities.

The search was performed in five online computer science libraries: ACM Digital Library[1], IEEE Explore[2], SpringerLink digital library[3], Wiley Online Library[4] and ScienceDirect[5]. The search was conducted using following general term, limiting the results to the field of agile methods, related to fields of software and security engineering:

```
agile AND software AND (security AND engineering)
```

The library specific searches and the size of each result set are presented in Table 1. ACM Digital Library search produced 59 results, IEEE Explore search 120, and SpringerLink 101 articles. Wiley search was conducted in all fields available for search, and produced 32 results, and ScienceDirect produced 64 results. Inclusion of terms ‘security’ AND ‘engineering’, adjusted for each search engine, was crucial in obtaining a manageable result set. After the initial search, the result sets were scanned and articles which did not explicitly concern an agile software development method with security engineering activities were dropped. SpringerLink and IEEE result sets required a careful manual screening, while ACM produced a readily more homogeneous result set. Five out of the 11 selected methods were selected from the ACM Digital Library. Three methods were selected from the Springer Link result set, two from the IEEE Explore and one from ScienceDirect. None were selected from Wiley.

After the library search, a manual screening process was performed. The manual selection process was executed in following steps:

(1) Article describes use of an agile method, or use of an extension to an existing agile practice
(2) Article describes use of security engineering activities in the agile setting

The selected articles were reviewed and analysed, and the specified security activities identified. At this point, some papers were discarded from the result set either due to lack of ability or lack of clear security engineering activities. These activities were evaluated using the principles of Common Criteria, which is the ISO standard to “permit comparability between the results of independent security evaluations. It does so by providing a common set of requirements for the security functions of IT products and systems and for assurance measures applied to them during a security evaluation” [10]. Furthermore, each method was screened for any empirical evidence provided, as were references to any security, quality or safety standard.

Citation count was hypothesized to project industry or scientific significance. However, the citation count of individual articles outside their respective research groups and direct affiliates was found to be very low. One paper [20], without any citations, was selected for its merit in summarizing CLASP security process from an agile point of view.

Based on the search result set it was observed that ASD, Crystal, DSDM, EVO, FDD, RUP and TDD appear to have been decreasing rapidly. No examples of these methods got selected into this study. XP is still a mainstay of the agile software methodologies, although Scrum-based methods have since gained more popularity. Kanban, despite its popularity, appears to be less used for security specific purposes. These observations echo the results from surveyed industry agile practises, such as [22] and [12].

The methods selected for this review are presented in Table 2.

4 REVIEWS

The found methods, listed also in Table 2, are presented in forthcoming subsections. For each method, we shortly summarize how it enhances the existing methods and is there any empirical evidence or support for the method. The methods are organized primarily by publication year, but by grouping articles from the same group of authors together.

[1] http://dl.acm.org
[2] http://ieeexplore.ieee.org
[3] http://link.springer.com/
[4] http://onlinelibrary.wiley.com
[5] http://www.sciencedirect.com/

Table 2: Studies selected for review

Topic summary	Standards	Empirical	Cited	Source
Security Engineering and XP	Yes	No	3	Wäyrynen et al. 2004 [23]
Towards Agile Security in Web Applications	No	No	5	Kongsli et al. 2006 [11]
XP Practices and Security Requirements Engineering	No	Yes	10	Boström et al. 2006 [5]
Agile Security Using an Incremental Security Architecture	No	No	4	Chivers et al. 2005 [6]
XP Security Practices	No	No	3	Ge et al. 2007 [9]
CLASP, SDL and Touchpoints compared	No	No	40	De Win et al. 2009 [7]
Integration of security activities into XP with	No	No	0	Sonia et al. 2014 [20]
Agile security management	No	No	6	Baca et al. 2011 [4]
Security Activities in Agile Projects	Yes	Yes	1	Ayalew et al. 2013 [2]
Extending agile methods with security	No	Yes	8	Othmane et al. 2014 [3]
Software Security Skills, Usage and Training Needs in Agile	Yes	Yes	1	Oyetoyan et al. 2016 [16]

We review the key contributions of each study primarily from the security activity point of view. Security activities were extracted and placed into an activity matrix, summary of which is presented in Table 3. In the few cases where the activities were found difficult to categorize, the categorization process is disclosed.

The activities were placed into a life cycle model, consisting of 6 phases: Pre-requirement, Requirement, Design, Implementation, Testing and Release. This generic lifecycle phase division is used in

4.1 Security Engineering and eXtreme Programming: An Impossible Marriage? [23]

The oldest of the selected articles by Wäyrynen, Bodén, and Boström [23] concentrates on extreme programming, and provides an analysis of the method from the viewpoint of two central ISO/IEC standards for software security: Systems Security Engineering – Capability Maturity Model (SSE-CMM, ISO 21817) and the Common Criteria (CC, ISO 15408). The article provides a solid analysis of the agile security: it discusses the topic of insecurity of agile methods, reviews the literature and earlier work aiming to integrate security activities into agile methods and transform them into agile activities, and then provides an analysis standing on two cornerstones of the traditional security engineering standards.

The issues discussed in this article from 2004 are echoed and reiterated in most of the later works over and over again: the agile methods' perceived unsuitability of security work, the contradicting requirements for fast and continuous delivery and meticulous design and planning, and the fundamental contradiction between security engineering (i.e. sequential development, formal processes, nonnegotiable requirements) and agile methods (i.e. incremental development, free-form work flow and low overhead, fluid goals and requirements). The authors set aside much of the prejudice, and concentrate in reviewing XP from a standardized security engineering perspective. XP is analysed from the viewpoint of SSE-CMM's requirement to specify security needs and Common Criteria's requirement to provide assurance that the said requirement is met.

The discussion of the topic is more on the philosophical rather than practical side, as there is no empiric evidence to test the hypotheses; the authors do, however, succeed in arguing their case that XP is no less well suited for security engineering activities than any other software development method. As a matter of fact, many of the XP's inherent properties are seen as beneficial to security engineering, such as simplicity of the design, pair programming, collective ownership of the code, test-driven development, refactoring and coding standards. Certain security activities are noted to be, if not incompatible with XP, at least missing from it: need for security engineer(s), static code review, security policies, formal security reviews and security documentation are mentioned. The suggested solution is as simple as one would assume: integrate the security engineering activities into the agile method.

The authors acknowledge the need for empiric validation missing from this article. They present a claim that their approach is limited to particular fields of software engineering, specifically excluding real-time software and safety critical applications. The paper is the earliest one of the selected studies, pointing out that certain agile practises are also beneficial security activities. The identified activities were security education, misuse cases, simplicity of (security) design, pair programming, and security testing. The security activities acknowledged by the authors to be missing from XP were not included in the results.

4.2 Extending XP Practices to Support Security Requirements Engineering [5]

The paper by Boström, Wäyrynen, Bodén, Beznosov, and Kruchten [5] enhances XP by introducing two security specific mechanisms: abuser stories as threat scenarios, and security-related user stories as security functionalities. The authors suggest that incorporating the security engineering activities into XP as agile processes, using XP terminology and mechanisms, yields the intended benefits without sacrificing the agile method's claimed benefits. The security-augmented XP process is explained starting from requirements planning, and introducing a way to integrate the abuser stories and security-related user stories into XP's planning game. Since stories are the essential way of communicating requirements and features in XP, the two new story types affect the whole development process.

The method is applied in a student project, which is described from the security point of view: specifically, the implications of the added features on the XP process are discussed. For the project, they have also introduced the role of a security engineer, which is typically an explicit requirement in various security standards. The

Common Criteria is mentioned several times, and its requirements discussed, yet the method or the described student project do not claim compliance with any security standard, nor specifically provide the security assurance that would satisfy the CC requirements. The proposed security activities consist of two key design phase processes: use of misuse cases (abuser stories, security-related user stories), which were considered to have effect when applying security principles to design.

4.3 Towards Agile Security in Web Applications [11]

The study by Kongsli [11] is one of the earlier efforts to integrate security engineering activities with agile software engineering methods. The paper reiterates the then-current strongly negative attitude towards the agile methods' ability to support security activities, assumed to result in less secure software products. The reasons for this are still basically unchanged, after a decade of effort: security methodologies are sequential and require 'big design up front'. The paper does not elaborate further mismatches, and from there on, concentrates on the security activities to be incorporated in agile methods. Use of XP is implied, yet not specifically stated; also Scrum is mentioned.

The article lists several agile security activities, collected from earlier literature: misuse stories (also called abuse stories) to supplement user stories; automated security tests throughout the development, parallel to unit and acceptance tests to verify the defence against misuse stories; security review meetings, comparable to iteration planning meetings, to provide the team with an overview of the security-related issues; and, finally, securing the deployment earlier than in sequential development, where system hardenings, security tests and security risk mitigation is done after the software functionality and features are completed. Authors then discuss the shortcomings, incompleteness of the misuse cases and tests, largely specific to the used testing tool, Selenium. They also discuss the role of security expert in software development, which transforms in agile process from owner of the security issues into a coach, who guides the team which collectively owns the misuse cases (stories) and the security in general.

The paper does not provide direct empiric validation for their method, yet implies that the method has been used in a customer project. There is no mention about a need to follow security regulations regarding the software or the development process. The amount of security activities mentioned in this study was quite low, but these can be considered key components in building secure software: misuse cases and security testing. Security review meeting was considered to be a form of verifying the security attributes of resources.

4.4 Agile Security Using an Incremental Security Architecture [6]

The article by Chivers, Paige, and Ge [6], as many of the other early works in the field of agile security engineering, starts by acknowledging the contrast between agile development, namely XP, and traditional security engineering approaches. The approach concentrates on the iterative nature of the agile methods, and also the inherent cost of refactoring, contrasted to maintaining a sufficient level of security. The authors propose an iterative security architecture which maintains 'good enough security' throughout the development iterations. The agile approach to architecture is described in Kent Beck's words as *the simplest thing that could possibly work*, and on the other hand admit that an architecture acts as a useful artifact for maintaining and communicating the overall vision of the system under development. This approach is used to define also the security architecture presented in this article. Good security architecture is stated to partition a system and identify its security-sensitive parts (and in what way), show how the security components combine for useful system level security, and communicate the structural security logic in an attempt to ensure that the development team does not build functionality that bypasses security.

Security architecture is partly approached from the refactoring point of view: architecture is written as the code is produced, and re-written during following iterations, to reflect the security view of the current system. The authors claim that the architecture should be just that: not a plan for future, but as clear and and simple representation of the current state as possible. This is logical from the agile point of view, which promotes avoiding top-down planning as the goals may change before they are reached. The case is presented as a 'paper exercise' where an iterative software project's architecture evolves during each iteration along with added functionality. The security context in the example project is user authentication scheme, with alternative architectures: presented alternatives are a heavy top-down design, which at certain stage will have to be abandoned and changed to another, the scenario where there is no architecture and the developers select the easiest way which soon becomes unmanageable; and, finally, there is the 'just right' architecture that also may have to be abandoned (of course, not in the scenario presented), but is much less expensive as it corresponds to the current need, not an anticipated one.

The article does not present empiric evidence, nor claim standard support. It does, however, provide a way to provide security assurance in the form of security architecture in a way that conforms with agile values. The security activities covered by an iterative security architecture are not explicitly stated in the description, yet the provided example conveys the idea that the author has had in mind. These were expanded to cover most of the requirement phase activities, and due to its iterative nature, also contain the most common design phase activities: documentation of assumptions, detailing misuse cases, applying security principles to design (in response to changing architecture-level requirements), and performing a security analysis of the system design.

4.5 Extreme Programming Security Practices [9]

Ge, Paige, Polack, and Brooke [9] propose security training and fundamental security architecture for XP as means to establish a security criteria. Security training would effect the team's way of conducting the XP's planning game: team's awareness of security issues would bring security issues into the center of the stage. Specifically, they would write security stories, understand security threats and vulnerabilities, be able to write better code and

avoid common mistakes, and be able to perform security testing. Fundamental security architecture, in contrast to an iterative security architecture, is defined before the iterations commence, and outlines the available security mechanisms, system platforms and provides basis for risk and vulnerability assessments. An example representation of a fundamental security architecture would be a collection of engineering patterns. The architecture could be created based on architectural risk analyses, security and programming best practises, and from both tacit and documented software project experience and knowledge.

The article does not provide empiric evidence, nor provide much more than the outlines of the proposed security activities. Both security training and and a system-level security architecture are, however, a common requirement in software security standards, and the authors show their validity for agile methods in the context of extreme programming (XP). Security training is a direct and straightforward security activity. Fundamental security architecture, as proposed in this article, does not specify the content of the architecture model itself. It was interpreted to cover most of the general pre-requirement and requirement phase activities specified in the then-current CLASP security process.

4.6 On the secure software development process: CLASP, SDL and Touchpoints compared [7]

De Win, Scandariato, Buyens, Grflgoire, and Joosen [7] have conducted an extensive and exhaustive review of three then-current industry standard security processes, OWASP CLASP, Microsoft SDL and Cigatel Touchpoints. The security activities in these processes are extracted, categorized and placed into an activity matrix, with a total of 153 distinct entries. Each security process is discussed and characterized, and their similarities and common properties pointed out. An theoretical case is presented, applying all three processes into an example development project.

The suitability of the processes with agile methods is not discussed, although the authors acknowledge the prevalence of the XP method, and note a knowledge gap in applying CLASP to agile methods, where SDL and Touchpoints are noted to be more extensive in this matter.

Although not directly discussed in the article, the security processes analyzed and compared are designed for basic compliance with ISO security standards (SSE-CMM and Common Criteria). The study presents an example case into which the security processes are applied, but not direct empirical evidence.

It should be noted that as this study aims to chart out all the security activities in several security processes, they are not listed as occurrences in the resultant table of security activities in this study.

4.7 Agile Development with Security Engineering Activities [4]

Baca and Carlsson [4] start by stating the existence of classic suspicion against the security of agile methods, naming the fast pace of development and perceived lack of documentation as the greatest discrepancy between the agile and secure engineering values.

The selected approach is to introduce and review Microsoft Security Development Lifecycle (SDL), Cigatel Touchpoints and the Common Criteria. The activities specified in these processes are then used to evaluate an industry agile software development method, enhanced with proposed security activities. Scrum and XP are used as reference methods for the iterative and incremental development process. The evaluation criteria for the security activities was to give the developers chance to rate the cost and benefit of each security activity.

The company where the study was conducted was traditionally not using iterative methods, which is represented by the roles present in the development process: the development team consisted not only of developers but also of testers, architects and requirement engineers. The feedback received from the development team, product owners and the unit manager contributed to selection of the most preferred and beneficial security activities. Product owners preferred security requirements from CC and role matrix from SDL. The development team preferred at design phase to use assumption documentation from Touchpoints, abuse cases from SDL, and requirements inspection from CC, added with countermeasure graphs. For implementation phase, the static code analyses and coding standards were seen most beneficial. Testers preferred dynamic analyses and, of course, testing.

The security activities ranking by the developers themselves produced interesting results, and, according to authors, selecting the activities based on feedback produced a flexible security engineering process that also conforms with agile principles. The developer opinion may well be biased, resulting compromised security: this is apparent in the project team's attitude towards fuzz testing: this was seen as costly and therefore non-beneficial, so it was not implemented at all. This very poorly reflects e.g. the Microsoft SDL's view on fuzz testing: in SDL, fuzz testing is seen as an important and easily automated part of security testing of various software interfaces. The article summarizes neatly the security activities suggested by existing security frameworks, giving a good overview of activities valid for standard compliance – which the method does not claim. The method and the security activities were not integrated into a real production project, they were merely tested and discussed in the existing sequential software project context.

Article presents a solid set of 11 security activities. Of these, the release-phase activity of repository improvement was not found on any of the other studies. This is done in retrospect, and may be considered to further promote continuous security between iterations.

4.8 Identification and Evaluation of Security Activities in Agile Projects [2]

Ayalew, Kidane, and Carlsson [2] have selected several security engineering activities from industry security frameworks: CLASP, SDL, Cigatel Touchpoints and the Common Criteria, and performed a survey-based cost-benefit analysis of these security activities. The approach extends the previous work by producing a list of 'agile compatible and beneficial security activities'. This resulting list of 16 activities appears to cover all phases of the secure software development life cycle. Countermeasure graphs, introduced by Baca and Carlsson in [4] are present also in this study, which, coupled with

the same the life cycle phase model, is another concrete evidence of the connection between these studies. The similarity between this and the previous studies is noted and discussed, and the extended results and marked differences of this paper are pointed out.

Inclusion of CLASP marks a concrete improvement in resultant set of agile security activities, especially in the release phase. Assigning a value for agility is not self-explanatory, as each organization may estimate the agility of an activity differently. Also, selecting security activities solely based on their agile value does not guarantee acceptable security. Combined with the other studies, the activities selected by this process provides a valuable addition to the set of agile security activities. This article lists several security activities extracted from SDL, CLASP and CC, and selects the most agile ones out of them. The resulting set of 16 security activities was further reduced for the purposes of this study: SDL's security tools and countermeasure graphs were removed, resulting in 14 key agile security activities, covering all phases of the security development.

4.9 FISA-XP: An Agile-based Integration of Security Activities with Extreme Programming [20]

This article by Sonia, Singhal, and Banati [20] represents the practise of assigning security activities an agile value, in this approach complemented by an online tool where the security activities can be chosen based on a user-assignable Acceptable Agility Reduction Factor. The security activities are derived from OWASP CLASP (Open Web Application Security Project, Comprehensive Lightweight Application Security Process). Microsoft's SDL is dismissed on account of an earlier study [7] branding it more 'heavyweight and rigorous' than CLASP, and therefore not suitable. Development of CLASP has since seen continuation in OWASP SAMM [15], while SDL has evolved towards a more agile-friendly adaptation [see 13, 14].

For the purposes of this study, this article works mostly as a summary of the [7] from an agile point of view, also filling the 'knowledge gap' applying CLASP to agile, identified in that study. An important contribution is mapping of XP's agile activities into the 30 listed security activities, although restricted to CLASP process activities. This mapping provides further evidence towards the conclusion that a fundamental mismatch between security engineering and agile methods does not exist: the authors provide an integration matrix (Table 2 in [20]), in which every single security activity is found basically compatible with at least one agile activity. A clear fault of this integration matrix is the mapping of several security activities into the agile activity of pair programming: for example, operational security guide is not a result of pair programming, and does not necessarily even take place at the implementation phase. This does not imply incompatibility between the activities, just a misconception and misplacement in the matrix. Despite these concerns, and after careful consideration, it was decided that this article will be included in this review.

Pre-assigning an agility value to security activities can prove useful in selection of security activities for an agile project, with specific needs to satisfy a security requirement and to provide security assurance. A requirement for compliance with a security standard would be a perfect example of such a selection criteria. Standard compliance is not discussed in this article. The article does not provide direct empiric evidence to support its agile value assignment technique. As with [7], the security activities listed in this article are not presented in the result matrix, Table 3.

4.10 Extending the Agile Development Process to Develop Acceptably Secure Software [3]

The article by b. Othmane, Angin, Weffers, and Bhargava [3] begins with a traditional comparison of iterative and incremental, i.e., agile software development methods and sequential security engineering activities and processes. These challenges are derived from literature, and listed as "lack of complete view of the system, absence of security engineering activities in the development process, lack of detailed documentation, lack of security awareness of the customers, and conflict of interests between security professionals and developers". The authors list certain earlier attempts to integrate security activities into agile methods, but criticize these for their lack of *continuous security*, calling for an agile development method, that produces acceptably secure software in each iteration.

The authors list several approaches to agile methods with security activities: OWASP and Microsoft security frameworks, and risk-driven and security assurance-driven software development methodologies. They proceed to proposing their own approach, which aims to continuous security and security assurance by adapting security activities and a specific security reassurance process into each increment and release. The security activities are listed per life cycle phase:

- Inception: threat modeling, risk estimation, and identification of security goals.
- Construction: defining security claims, writing security stories, and defining the security assurance. All of these are done for each iteration.
- Transition: performing the security assurance tasks, for example producing the documentation artifacts, running (automated) security tests, or conducting an external review (security audit).

This article presents a methodology which not merely integrates security activities into an agile method, but also offers a method to provide continuous security assurance. Although somewhat limited and without concrete empiric evidence, each incremental release candidate produced by applying the proposed is secure and ready for acceptance. Although complying with security standards is not specifically mentioned in the text, this approach includes all the necessary components for standard compliance. A real world approach might not aim for continuous security assurance, i.e., transition phase tasks done during each iteration, which could improve the approach by reducing the workload and therefore the cost of security tasks. On the other hand, continuous security assurance makes the proposed method directly eligible for DevOps model, with frequent delivery of new software increments into production.

4.11 An Empirical Study on the Relationship between Software Security Skills, Usage and Training Needs in Agile Settings [16]

In this case study by Oyetoyan, Cruzes, and Jaatun [16], the security activities of two organizations using agile software development methods are inspected and compared. The study was conducted by combining 26 security activities from CLASP, SDL, Touchpoints and the Common Criteria, and asking the development teams which of the activities does their organization employ. The focus was in the skill, training and experience of the development teams in each of these activities. Both surveyed organizations used Scrum as their agile software development method. The researchers had selected four common agile activities as 'frequently used activities': use of a code review tool, static code analysis, use of a static code analysis tool, and pair programming. Promoting pair programming into a core activity used by both of the surveyed organizations is somewhat atypical, as a wider industry study finds pair programming among one of the least utilized practises [12]. Also, unlike in XP, pair programming is not a key technique in Scrum. Additionally the security activities were categorized into two groups: core activities and activities that can be leveraged to deliver security.

The key findings from security activity perspective are difficult to generalize. It does appear, however, that largely regardless of the selected key activities in an organization, awareness of security issues and security training promotes their use – which was exactly what was hypothesized. Also, the security awareness in the form of training should be administered to all participants in the software development process: architects, developers and testers, to yield the best results. The ultimate drivers for a more frequent and thorough use of security activities cannot be deducted from the results, but the presence of an organizational security expert group in one of the organizations is listed as a potential source of promotion of security activities. The presented security activities are selected from industry-proven security processes.

The study is empirical and surveys existing organizational practises. Of security standards, the Payment Card Industry Data Security Standard (PCI-DSS) is specifically mentioned, and others hinted at. The presence of a security expert team and the listed security activities – secure design and coding, threat modeling and risk management, security testing, and security considerations at requirement and release phases – are all among to common security activities performed when meeting standard requirements [see e.g. 18].

5 RESULTS

The purpose of this study was to outline and identify the security activities used in agile software development. The papers selected for this study represent the whole development life cycle, from the project inception to the release. Secure DevOps delivery model was not in the central focus of the study, but was still found to be directly supported by at least one of the proposed methods. Our key finding can be summarized in the secure agile development life cycle: every phase of a software development project has been effectively covered by at least one agile security activity, including iterative architectures along the multitude of activities used in the development and release phases.

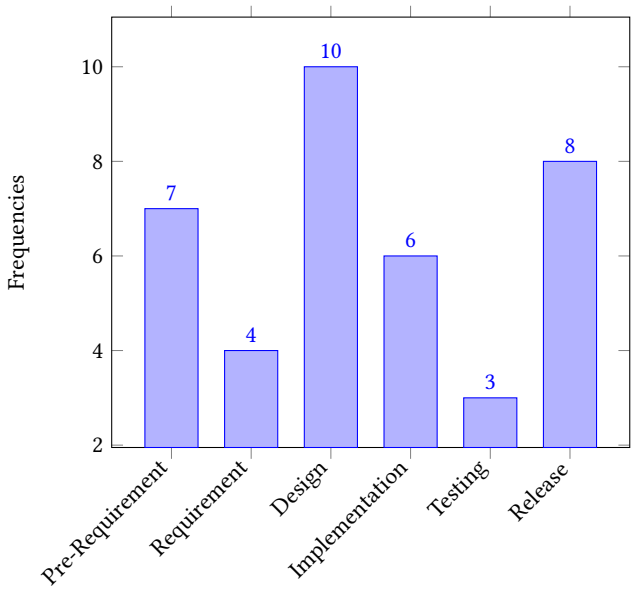

Figure 1: Distribution of the activities by life cycle phase

An adjusted total of 38 individual security activities were identified in these studies, of which 34 were present in more than one case. Even the least used activities have their uses: for example, while nominating a security officer or using a separate 'red team' for security testing can not be considered agile at all, they represent established industry processes and provide intrinsic value for the security process. Some distinctively non-agile activities may additionally be required by security regulation and therefore accepted as a form of customer requirement.

The security activities presented in each study were extracted and placed in the rudimentary life cycle model presented in [4] and [2], which in turn closely represents the phases used in Microsoft SDL. Some of the activities were grouped into main activities, and some more generic activities, such as the iterative security architecture in [6] were considered to contribute into several activities in the requirement and design phases. The resulting set of activities and the coverage of the agile security development life cycle is presented in Figure 1.

An overview of the gathered security activities is presented in Table 3. Certain activities, such as fuzz testing and red team testing, were combined under other activities, in these cases under 'Perform SW security fault injection testing' and the generic 'Identify and implement security tests', respectively.

A majority of the actions, 30 of 38, were selected from CLASP as presented and adapted to agile in [20]. Corresponding SDL, Common Criteria and Touchpoints activities are combined under the definition provided in CLASP. Selection of the security activities should always based on the task at hand rather than a popularity contest, yet the number of their uses was recorded. A low number of occurrences, 0 or 1, suggests a more uncommon adaptation of the activity, and a higher number can be interpreted as an indication of wider adoption. The top activity list of security actions looks as follows; in case of a tie, all the most used activities are listed.

Table 3: Security activities in life cycle phases

Phase	Activity	Occurrences	Source(s)
Pre-requirement			
	Specify operational environment	1	[9]
	Identify global security policy	1	[9]
	Institute security awareness program	3	[9], [23], [2]
	Monitor security practises	1	[9]
	Research and assess security solutions	1	[9]
	Build information labeling scheme	1	[9]
	Project security officer	1	[16]
	Pre-Requirement phase adaptations total	9	
Requirement			
	Identify user roles and requirements	5	[4], [6], [9], [16], [2]
	Perform security analysis of requirements	5	[4], [6], [9], [16], [2]
	Specify resource-based security properties	2	[6], [16]
	Requirement inspection	1	[4]
	Requirement phase adaptations total	13	
Design			
	Risk estimation	3	[3], [16], [2]
	Threat modeling	2	[3], [16]
	Document security design assumptions	5	[4], [6], [3], [16], [2]
	Detail misuse cases	7	[11], [4], [5], [6], [23], [3], [16]
	Apply security principles to design	4	[5], [6], [23], [2]
	Specify DB security configuration	0	
	Perform security analysis of system design	3	[4], [6], [2]
	Design UI for security functionality	0	
	Annotate class designs with security properties	1	[16]
	Quality gates	1	[2]
	Design phase adaptations total	26	
Implementation			
	Coding standards	3	[4], [16], [2]
	Pair programming	2	[23], [16]
	Integrate security analysis into build process	1	[2]
	Implement and elaborate resource policies	0	
	Implement interface contracts	0	
	Address reported security issues	1	[16]
	Implementation phase adaptations total	7	
Testing			
	Identify and implement security tests	6	[11], [4], [23], [3], [2]
	Perform security function usability testing	1	[4]
	Perform SW security fault injection testing	1	[2]
	Testing phase adaptations total	8	
Release			
	Manage system security authorization agreement	1	[3]
	Perform source level security reviews	3	[4], [3], [16]
	Verify security attributes of resources	1	[3]
	Manage certification process	1	[3]
	Perform code signing	2	[3], [2]
	Build operational security guide	2	[3], [2]
	Manage security issue disclosure process	3	[11], [3], [16]
	Repository improvement	1	[4]
	Release phase adaptations total	14	
	Adjusted total number of security activities	38	

- Pre-requirement: Institute security awareness program (security training)
- Requirement: Identify user roles and requirements (role matrix), Perform security analysis of requirements
- Design: Detail misuse cases (abuser and security stories)
- Implementation: Coding standards
- Testing: Identify and implement security tests
- Release: Perform source level security reviews (static code reviews)

Security activities in the design phase appear especially widely adapted to agile development, with a total of 26 documented implementations. The placement of abuser and/or misuse stories were present majority of the (in 8 of 11). Although some sources bucketed all stories into the implementation phase, they were considered to belong to the design or planning phase of each iteration or sprint. Pre-requirement phase had 9 implementations, while there were 16 found in the requirement phase. Implementation and testing phases are more tool oriented, but had still 11 and 13 implemented activities across the analyzed studies. Release phase, with static code reviews and strong emphasis on *security assurance* tasks, had 21 total security activity implementations. In total, the whole cycle of software development process appears well covered.

6 DISCUSSION

The findings implicate that while adoption of agile software development methods has been perhaps slower in the security field, but nevertheless it has largely already happened. The perceived discrepancy between agile values and security requirements has largely been solved, and both security activities and agile methods – and the experts utilizing them – have adapted to their integrated use. Our intention was not to inspect the agile security 'myth', yet the theme occurred in so many papers, especially in the earlier ones, that we decided the time has come to officially declare the death of of the incompatibility myth.

Certain limitations are acknowledged in the selected approach: the articles were selected from the result set in common digital libraries, and the source material is far from complete. Notable work with further – or contrary – evidence may not have been included in the search. The invariably positive view of agile secure activities in all of the surveyed articles can also be a reflection of publication bias: cases, where agile methods are found not to be adaptable simply do not get published, as they do not have a scientific or even novelty value. Furthermore, certain types of the software products and organizations may provide easier access to empiric evidence.

From strictly agile security engineering point of view, the evidence gathered by this study appears bipartite: theoretical models support the use of security engineering activities in agile software development, whereas empirical evidence is sporadic and largely incomplete. Also, any concrete security or quality metrics collected from the use of these activities appear to be missing. No examples of properly documented cases of agile security engineering in a security standard regulated setting were included in the result set. While alarming, this can also be attest either publication bias, or simply the incompleteness of the source material included in the selected digital libraries.

7 CONCLUSIONS AND FUTURE WORK

The practise of security engineering is well adapted to and widely used with agile software development methods. Some of the activities have been modified to better suit iterative development model, and, based on the literature, much attention has been paid to retain the agile nature of the development process despite of the added security activities. The often-repeated myth of agile methods' incompatibility or inherent unsuitability for security tasks, has been effectively been broken already in the first observed study, published in 2004. Yet the myth is perpetuated for years after this, even by the same authors, well up until mid-2010's. The field of secure agile software development is still fragmented and organization specific, largely due to highly adaptable nature of the agile methods. A wide industry survey concentrating on agile security activities would help identifying the key agile security practises and confirm the findings outlined in this study. Also, a distinctive similarity of security and safety activities implies an increase in the general quality of the software products created with security-augmented processes – a hypothesis that should be further inspected with supporting empiric evidence.

REFERENCES

[1] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. 2002. *Agile software development methods - Review and analysis.* Technical Report 478. VTT PUBLICATIONS.

[2] Tigist Ayalew, Tigist Kidane, and Bengt Carlsson. 2013. *Identification and Evaluation of Security Activities in Agile Projects.* Springer Berlin Heidelberg, Berlin, Heidelberg, 139–153. DOI: http://dx.doi.org/10.1007/978-3-642-41488-6_10

[3] L. b. Othmane, P. Angin, H. Weffers, and B. Bhargava. 2014. Extending the Agile Development Process to Develop Acceptably Secure Software. *IEEE Transactions on Dependable and Secure Computing* 11, 6 (Nov 2014), 497–509. DOI: http://dx.doi.org/10.1109/TDSC.2014.2298011

[4] Dejan Baca and Bengt Carlsson. 2011. Agile Development with Security Engineering Activities. In *Proceedings of the 2011 International Conference on Software and Systems Process (ICSSP '11).* ACM, New York, NY, USA, 149–158. DOI: http://dx.doi.org/10.1145/1987875.1987900

[5] Gustav Boström, Jaana Wäyrynen, Marine Bodén, Konstantin Beznosov, and Philippe Kruchten. 2006. Extending XP Practices to Support Security Requirements Engineering. In *Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems (SESS '06).* ACM, New York, NY, USA, 11–18. DOI: http://dx.doi.org/10.1145/1137627.1137631

[6] Howard Chivers, Richard F. Paige, and Xiaocheng Ge. 2005. *Agile Security Using an Incremental Security Architecture.* Springer Berlin Heidelberg, Berlin, Heidelberg, 57–65. DOI: http://dx.doi.org/10.1007/11499053_7

[7] Bart De Win, Riccardo Scandariato, Koen Buyens, Johan Grflgoire, and Wouter Joosen. 2009. On the secure software development process: CLASP, SDL and Touchpoints compared. *Information and Software Technology* 51, 7 (2009), 1152 – 1171. DOI: http://dx.doi.org/10.1016/j.infsof.2008.01.010 Special Section: Software Engineering for Secure SystemsSoftware Engineering for Secure Systems.

[8] Tore Dybå and Torgeir Dingsøyr. 2008. Empirical Studies of Agile Software Development: A Systematic Review. *Inf. Softw. Technol.* 50, 9-10 (Aug. 2008), 833–859. DOI: http://dx.doi.org/10.1016/j.infsof.2008.01.006

[9] Xiaocheng Ge, RichardF. Paige, Fiona Polack, and Phil Brooke. 2007. Extreme Programming Security Practices. In *Agile Processes in Software Engineering and Extreme Programming*, Giulio Concas, Ernesto Damiani, Marco Scotto, and Giancarlo Succi (Eds.). Lecture Notes in Computer Science, Vol. 4536. Springer Berlin Heidelberg, 226–230. DOI: http://dx.doi.org/10.1007/978-3-540-73101-6_42

[10] ISO/IEC. 2014. Information technology - Security techniques - Evaluation criteria for IT security ISO/IEC 15408:2008. (2014).

[11] Vidar Kongsli. 2006. Towards Agile Security in Web Applications. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '06).* ACM, New York, NY, USA, 805–808. DOI: http://dx.doi.org/10.1145/1176617.1176727

[12] Sherlock Licorish, Johannes Holvitie, Rodrigo Spinola, Sami Hyrynsalmi, Jim Buchan, Thiago Mendes, Steve MacDonnell, and Ville Leppänen. 2016. Adoption and Suitability of Software Development Methods and Practices - Results from a Multi-National Industry Practitioner Survey. In *2016 Asia-Pacific Software Engineering Conference (APSEC).* IEEE.

[13] Microsoft. 2017. Agile Development Using Microsoft Security Development Lifecycle. (2017).

[14] Microsoft. 2017. Security Development Lifecycle for Agile Development. (2017).

[15] OWASP. 2017. Software Assurance Maturity Model. (2017).

[16] T. D. Oyetoyan, D. S. Cruzes, and M. G. Jaatun. 2016. An Empirical Study on the Relationship between Software Security Skills, Usage and Training Needs in Agile Settings. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*. 548–555. DOI: http://dx.doi.org/10.1109/ARES.2016.103

[17] Kalle Rindell, Sami Hyrynsalmi, and Ville Leppänen. Case Study of Agile Security Engineering: Building Identity Management for a Government Agency. *International Journal of Secure Software Engineering* 8 (????), 43–57. Issue 1.

[18] Kalle Rindell, Sami Hyrynsalmi, and Ville Leppänen. 2015. A Comparison of Security Assurance Support of Agile Software Development Methods. In *Proceedings of the 16th International Conference on Computer Systems and Technologies (CompSysTech '15)*. ACM, New York, NY, USA, 61–68. DOI: http://dx.doi.org/10.1145/2812428.2812431

[19] Kalle Rindell, Sami Hyrynsalmi, and Ville Leppänen. 2015. Securing Scrum for VAHTI. In *Proceedings of 14th Symposium on Programming Languages and Software Tools*, Jyrki Nummenmaa, Outi Sievi-Korte, and Erkki Mäkinen (Eds.). University of Tampere, Tampere, Finland, 236–250. DOI: http://dx.doi.org/10.13140/RG.2.1.4660.2964

[20] Sonia, Archana Singhal, and Hema Banati. 2014. FISA-XP: An Agile-based Integration of Security Activities with Extreme Programming. *SIGSOFT Softw. Eng. Notes* 39, 3 (June 2014), 1–14. DOI: http://dx.doi.org/10.1145/2597716.2597728

[21] Michael Unterkalmsteiner, Pekka Abrahamsson, XiaoFeng Wang, Anh Nguyen-Duc, Syed Shah, Sohaib Shahid Bajwa, Guido H. Baltes, Kieran Conboy, Eoin Cullina, Denis Dennehy, Henry Edison, Carlos Fernandez-Sanchez, Juan Garbajosa, Tony Gorschek, Eriks Klotins, Laura Hokkanen, Fabio Kon, Ilaria Lunesu, Michele Marchesi, Lorraine Morgan, Markku Oivo, Christoph Selig, Pertti Seppänen, Roger Sweetman, Pasi Tyrväinen, Christina Ungerer, and Agustin Yage. 2016. Software Startups – A Research Agenda. *e-Informatica Software Engineering Journal* 10, 1 (2016), 89–124. DOI: http://dx.doi.org/10.5277/e-Inf160105

[22] VersionOne. 2017. 11th Annual State of Agile Survey. (2017).

[23] Jaana Wäyrynen, Marine Bodén, and Gustav Boström. 2004. *Security Engineering and eXtreme Programming: An Impossible Marriage?* Springer Berlin Heidelberg, Berlin, Heidelberg, 117–128. DOI: http://dx.doi.org/10.1007/978-3-540-27777-4_12