Aligning Security Objectives With Agile Software Development

Kalle Rindell
University of Turku
Turku, Finland
kalle.rindell@utu.fi

Sami Hyrynsalmi
Tampere University of Technology
Pori 305, Finland
sami.hyrynsalmi@tut.fi

Ville Leppänen
University of Turku
Turku, Finland
ville.leppanen@utu.fi

ABSTRACT

Success of the software development process depends on its ability to transform its objectives into requirements, and implementing these into features and functionality. Security objectives in software development are increasingly converging with the business objectives, as requirements for privacy and the cost of security incidents call for more dependable software products. Development of secure software is accomplished by augmenting the software development process with specific security engineering activities. Security engineering, in contrast to the iterative and incremental software development processes, is characterized by sequential life cycle models: the security objectives are thus to be achieved by an approach in apparent conflict with the unaugmented software development processes. In this study, to identify the incompatibilities between the approaches, the security engineering activities from Microsoft SDL, the ISO Common Criteria and OWASP SAMM security engineering models are mapped into common agile software development processes, practises and artifacts. The organizational and technical aspects of the mapping are considered primarily from the point of view of achieving the security objectives set for the software engineering process: setting security requirements for design, the implementation of the security architecture and design, and the required security verification before releasing secure software through efficient software security development process towards secure software maintenance.

KEYWORDS

agile, software engineering, security engineering, methodologies

ACM Reference Format:
Kalle Rindell, Sami Hyrynsalmi, and Ville Leppänen. 2018. Aligning Security Objectives With Agile Software Development. In *Proceedings of ACM Conference (SecSE 2018)*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Software development organizations are hard pressed to meet the increasing demand for secure software [10, 15, 33]. Value-driven software development processes are seen lacking in ability to produce secure software, essentially a risk-based process. Responsibility for software security is placed on elements external to the development teams [6], deepening the separation of business objectives and security objectives in software development. In agile development, the lessened emphasis to preliminary planning, and the absence of fixed milestones causes difficulties incorporating external security processes into the iterative development processes: organizations effectively end up running a non-agile security development life cycle along the agile software development processes. Aligning the business and security objectives, and aligning and integrating the activities is necessary to avoid sacrificing neither efficiency of the agile processes, nor the long-term security objectives.

Agile software development processes call for agile organization, infrastructure and business models [4]. Self-organizing teams and non-deterministic implementation processes result in task implementation patterns remarkably different from those produced by sequential and pre-planned counterparts of these models. In addition to the organizational dissimilarities, security engineering processes are ultimately driven by *risk* rather than *business value*; unlike the agile development processes, they also rely on planned activities executed in a sequence [18, 38]. Deterministic sequences aim to reduce the security risk by executing pre-planned tasks at fixed points in the development life cycle. Lightweight, iterative, and incremental processes utilize a profoundly differently structured implementation and verification cycle; thus, security mechanisms fully integrated into agile development processes are required. There exists no inherent obstacle to utilizing agile processes to achieve the security objectives: implement the required security functionality and security assurance, and verify the absence of known security vulnerabilities [cf. 31].

The differences between the methodologies have been categorized: the methods are determined to be either risk-driven or value-driven [9]; hybrid models, such as Disciplined Agile Delivery [1], set out to reintroduce a set of planned activities (a sequential element) into the iterative work flow. To find out the reasons for the difficulties experienced by the software and security engineers, software security processes must first be defined, and the activities analysed. These differences between the approaches, values and even the paradigms of software engineering and system engineering methodologies lead to the primary research question:

RQ: How can the agile practises be integrated with software security engineering activities?

This question is considered primarily by mapping a set of agile agile practises, activities and artifacts into security development lifecycle phases; this mapping is preformed per each phase separately. In Chapter 2, the issues in software security and the current adaptation of agile software security engineering activities, practises and artifacts are examined. In Chapter 3, an exhaustive list of common software security activities are mapped into agile practises, processes and artifacts found common in the software development

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SecSE 2018, May 25, 2018, Porto, Portugal
© 2018 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

industry[21]; the security engineering activities defined in the Microsoft SDL security development lifecycle model, the ISO/IEC Common Criteria, and OWASP SAMM are used. In Chapter 4, the process and the related issues are discussed in the perspective of achieving both the security objectives and the business objectives of software development process; Chapter 5 concludes the article.

2 BACKGROUND

Software Security Engineering (SSE) introduces several system engineering practises and activities into the software development process. In academia, software engineering as a subdiscipline of computer science tends to systematically exclude unquantifiable variables as 'user' and 'operating environment' from its core [see 14]. However, in practice it is clear that in order to meet the software engineering's objectives of delivering working software in sustainable manner, software engineering and system engineering will have to meet [7]. Mainstream software development methods are extremely value-focused, and as such perform poorly when facing *non-functional* requirements [28]. Functional requirements describe *what* the system should do, whereas non-functional or qualitative requirements are typically worded as *how* the system should perform, or concern the architecture, operating environment, scalability etc. Treating security as a non-functional requirement has provided a convenient argument against the agile methods suitability for security engineering work [36], and even suggestions that agile methods are inherently ill-suited to produce secure software [cf. 29].

Security standards are guidelines for security implementation, and several international software security regulation frameworks exist. The Common Criteria [19] has been developed to quantitatively evaluate security. It contains concrete instructions and requirements for security functionality, and suggests a framework of *security objectives*, to be elicited into *security requirements*. The objectives are also used as a basis for security risk management process, and form the outlines of the software system's *security policies*. Security policies are implemented by a set of security activities and result in a phletora of functionalities which are verified by security testing and other verification methods. Some of the security activities bear a notable similarity with software quality assurance activities: these include code analyses and reviews, verification documentation and formal verification audits performed by a external certifying entity. However, treating security requirements categorically as non-functional reflects an insufficient understanding of what software security is, and how it is implemented in the software, and clearly departs from the security models provided by the ISO/IEC standards [19, 20].

2.1 SSE models in an agile context

Even though agile adaptations exist, SSE is predominantly performed by applying sequential models: however, security development, i.e., incorporating security functionality into the software is an implementation task among others. In practise, however, the security process is a formal review at a fixed point in time, not a continual process truly incorporated into the software development process. Injecting inspection points into the agile work flow necessarily requires pre-planning and thus has the potential to disrupt the goals of value-driven processes.

At least two common maturity models address also development-time information security issues: (1) The Software Security Engineering Capability Maturity Model (SSE-CMM), the ISO/IEC-standardized heavily process-oriented security management, metrics and implementation framework [20]. This model originates from the Capability Maturity Model Integration, developed by the Carnegie Mellon University and currently maintained by the CMMI Institute [22]. As a process model, applying the CMM-based model can be very costly [13], and can be projected not be limited to security improvement only. (2) The Open Web Alliance Security Project's (OWASP) open source licensed Software Assurance Maturity Model (SAMM) contains also development-time activities, and sets best practises for security governance, construction, verification and operations. OWASP has previously maintained also a model specific to software development, the Comprehensive, Lightweight Application Security Process (CLASP); this model has fallen out of common use and replaced by the SAMM. SAMM also bears distinct similarities with Building Security In Maturity Model (BSIMM) [34]. The BSIMM does not claim to specify security models or frameworks; it is published annually as a list of industry's state of information security, surveying the current best practises in security engineering.

The Open Web Alliance Security Project's (OWASP) open source licensed Software Assurance Maturity Model (SAMM) contains also development-time activities, and sets best practises for security governance, construction, verification and operations. OWASP has previously maintained also a model specific to software development, the Comprehensive, Lightweight Application Security Process (CLASP); this model has fallen out of common use and replaced by the SAMM. SAMM also bears distinct similarities with Building Security In Maturity Model (BSIMM) [34]. The BSIMM does not claim to specify security models or frameworks; it is published annually as a list of industry's state of information security, surveying the current best practises in security engineering.

The SAMM, combined with OWASP's implementation guidelines such as the OWASP Top 10 Application Security Risks [25], offers guidelines to build a security framework, complete with governance and security metrics. As a framework the SAMM follows a proven path: security strategy includes governance and metrics and enabled by security education; this can be considered to be equivalent of the Common Criteria's Security Target. Security threat assessment leads to security requirements, which form the basis for security architecture. Security design and implementation (code and software resources) are verified through analyses and reviews. Security testing is thoroughly addressed, and this stage contains also release criteria for the maintenance phase; this phase contains issue management, environment hardenings and 'Operational Enablement', providing instructions for secure DevOps (or, DevSecOps). The SAMM is divided into three maturity levels, each with specific objectives, activities, assessment criteria and expected results. These are discussed for each software security development lifecycle phase.

Microsoft SDL is an example of a Software Security Development Lifecycle (SSDL) model, representing the one of the several software

security formalization efforts initiated in mid-2000s. Other SSDL models include Touchpoints SDLC (Security Development Lifecycle, now part of BSIMM) and the OWASP CLASP, continued as SAMM.

2.2 Research description

Producing secure software by introducing security engineering processes and activities to an iterative and incremental software engineering process is challenging. To reverse this, the various phases and activities in software security development are extracted from the Microsoft SDL and the ISO Common Criteria. No specific agile methodology is used as a reference, but rather agile processes, activities and artifacts, which are linked to the security activities. Some of the terminology is derived from the Scrum method [32], currently the most commonly used mainstream software development methodology [30, 37]. The development lifecycle is divided into six phases, and the relevant security development activities are set into agile context. Positive and negative effects are then analyzed from the viewpoint of *achieving security objectives*. The concept of security objectives is derived from the Common Criteria, and visualized in Figure 1.

The Common criteria provides a framework for evaluating the security of a software-intensive product by setting a rather complex framework. The Security Target consists of the security measures for the software itself (Target Of Evaluation, TOE) and the operating environment; for the purposes of this study, only the security objectives of the TOE are considered. Security objectives are met by eliciting the security requirements, resulting in security specification which guides the implementation of security functionality. Some of the security functionality exists to explicitly provide security assurance. Security assurance, such as logs, verifies the existence and effectiveness of the security functionality implemented into the system. It also works as the basis for security metrics and helps tracking down the potential security breaches later in the software's life cycle. The Common Criteria provides ten example software security activities, which are used together with activities from SDL and SAMM.

As the software security development life cycle models are divided into distinct phases, the research is listed here in relation to the life cycle model. The life cycle models phases used are *pre-requirement*, *requirement*, *design*, *implementation*, *verification* and *release*. This model is in close resemblance of the SDL's model, omitting the maintenance phase, and has been used in previous studies by e.g. Baca and Carlsson [3] and Ayalew et al. [2]. The Common Criteria does not explicitly address the pre-requirement phase, but that is implied to consider the setting of the Security Target and the Security Objectives.

The agile practises, process and artifacts, into which the security activities are mapped, are derived from common agile methodologies. These are presented in Table 1.

The upper part of Table 1 lists the agile processes and process artifact; these can be considered the 'core' of agile development. The lower part, under the horizontal line, contains the software development practises associated with various agile methodologies, such as Scrum and Extreme Programming (XP). The 'Usage' column ranks the activity by the reported average usage.

Table 1: Selected agile practises, processes and artifacts and their use as reported in [21]

Code	Agile processes and artifacts	Usage
A1	Iterations	84.2%
A2	Iteration planning meetings	76.6%
A3	Iteration backlog	75.5%
A4	Product backlog	76.1%
A5	Daily meetings	69.6%
A6	Iteration reviews/retrospectives	72.3%
Agile practises	Usage	
AP1	Coding standards	81.2%
AP2	Test-driven development (TDD)	75.0%*
AP3	Simple design	74.5%
AP4	Continuous integration	73.9%
AP5	Refactoring	73.9%
AP6	On-site customer	49.5%
AP7	Pair programming	45.1%
AP8	Planning game	27.7%

* Value separately calculated from the result data.

The source survey for Table 1 has also been reported as agile practises' significance in reducing technical debt [17]. Managing technical deficiencies and recognized debt holds remarkable similarities to security engineering: many of the issues reported in this study, such as inadequacy of the architecture, structure, testing and documentation, are directly applicable to security work. In contrast, the actual features, requirements and defects represent a minority of the concerns for technical debt among the respondents, while these are central considerations in security work. The agile processes, practises and artifacts are mapped into software security development lifecycle phases in Figure 2 in Chapter 4: the security activities here are aligned to the activities in Table 1 by the SDLC phases as shown in Figure 2.

3 SECURITY ACTIVITIES IN AGILE SOFTWARE DEVELOPMENT PROCESS

This section lists all the security activities extracted from Microsoft SDL [18, 23], The Common Criteria [12, 19] and OWASP SAMM [24]. The activities for each lifecycle phase are examined and their adaptability to agile development and a matching agile activity are presented. The mapping presented in the following chapter is derived from literature individually for each practise; to the knowledge of the writers, an empiric or systematic framework for a direct mapping does not exist. For reference,

3.1 Pre-requirement phase

The security activities in this phase are presented in Table 2. The pre-requirement phase in the SDL contains only one item: core security training. For the purposes of agile development, this part should also contain training in the agile methods: processes, activities, tools, communication procedures, terminology and other indoctrination. Even good software engineers may be unaware of security issues and have a poor understanding of agile models; security engineers participating in software projects should be made

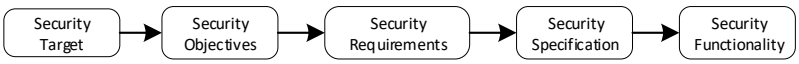

Figure 1: Simplified software security target framework (adapted from the Common Criteria [19]).

aware how mainstream software projects are conducted and what is their work flow.

Before the projects begin, SAMM suggests building security development strategy and roadmap, measuring the relative value of data and software assets, and establishing security and cost metrics; after these, the security expenditure can be assessed. SAMM also calls for establishment of security policies and security compliance; third and from the development process point of view the directly most important category at this stage is security education and guidance for all people relevant for the software and security processes. SAMM also places threat assessment onto level of organizational practises rather than project specific; a well-build organizational threat assessment framework should be able to address project-level security issues as well, and create the necessary input for security requirement gathering. At the third maturity level, SAMM advises to develop and deploy compensating controls for the threats. The threat assessment phase also provides the risk lists to be assessed in the security development projects.

In the CC framework, personnel trained in security is a part of the overall Security Target. Security training of the individuals is one of the Security Objectives to be fulfilled before the project begins. Other security objectives are typically application dependent and more difficult to generalize. However, skill is an universal requirement in both software development [8] as well as security engineering [see e.g. 26].

As this phase precedes the initiation of the development process, there are no directly applicable agile practises: communication with the on-site customer (AP6) may be started already at this point, and the security items added into a rudimentary version of product backlog (A4). Security training is an essential prerequisite, as skill and knowledge forms the base for all security engineering work [cf. 26].

3.2 Requirement phase

Requirement phase activities, presented in Table 3, contain activities necessary for security requirement elicitation. The SDL deals with the requirement phase activities at quite high level, and does not provide concrete resources, guidance or tools to perform them. SDL also suggests that security requirements and risks are defined only once in the project, although it is doubtful that they will remain static throughout the project. SDL's approach of setting quality gates is hardly security specific at all, yet this is something that can be addressed through agile practises of Coding Standards. The Common Criteria also stays at rather abstract level, and defines two types of security activities: definition of Security Functional Requirements (SFR) and Security Assurance Requirements (SAR). In the ISO standardization framework fulfilling both these requirement types is essential in achieving the security objectives, and verifying this.

SAMM's security requirement activities give a logical process how to gather and elicit security requirements: partially it relies on the business requirements, which are then evaluated against the compliance guidance for security requirements; this has been created in the pre-requirement phase. At the second level, an access control matrix is created, and the security risk list from previous phase used to complement the list of security requirements. At third level, SAMM calls for business-oriented security activities of security management for supplier contracts, and an audit program for security requirements.

Agile software development is all about *change*. Effectively this means efficient and continual requirement management. In Table 1 the only agile activity directly addressing requirement elicitation and prioritization is the Planning Game [5]. With 27.7 % adoption rate this technique, originating from the XP methodology, sets an example how requirement elicitation is done iteratively. Security requirement elicitation techniques have been surveyed by Tondel et al. [35], although this study does not address the issue specifically from agile software developer's point of view.

Requirement elicitation process must be thorough and systematically identify all the relevant security functionality and assurance requirements; iterative approach (A1) directly supports this process. Agile methods are extremely efficient in prioritizing the implementation queue: identified items are given workload or complexity estimates, and are then placed into the product backlog (A4). Work items will also get assigned an explicit Definition of Done (DoD). Eventually, depending on their priority, they will be picked up for the iteration backlog, get implemented and verified. Quality of the requirements is typically ensured through rigorous validation process: methods, such as INVEST for user stories (natural language requirements) and SMART for backlog items [39] are used to review

Table 2: Pre-requirement phase activities

Source	Security activity	Agile activities
SDL	Core Security Training	AP6
CC	Set security target and objectives	A4, AP6
SAMM	Strategy & metrics	
SAMM	Policy & compliance	
SAMM	Education & guidance	
SAMM	Threat assessment	A4, AP6

Table 3: Requirement phase activities

Source	Security activity	Agile activities
SDL	Establish Security Requirements	A1, A4
SDL	Create Quality Gates/Bug Bars	A1
SDL	Perform Security and Privacy Risk Assessments	A1, A4
CC	Definition of SFR and SAR	A1, A4
SAMM	Security requirements	A1, A4

Table 4: Design phase activities

Source	Security activity	Agile activity
SDL	Establish Design Requirements	A1, A3, A4, AP3, AP6
SDL	Perform Attack Surface Analysis/Reduction	A1, AP3
SDL	Use Threat Modelling	A1, AP3
CC	Cross-analysis of TOE designs	A1, AP3
CC	Vulnerability analysis and flaw hypothesis	A1, AP3
CC	TOE design analysis against the requirements	A1, AP3
SAMM	Security architecture	A1, AP3, AP6
SAMM	Design review	A1, AP3

the requirements and transform them into implementable features and functionality.

3.3 Design phase

Design-time activities are listed in Table 4. Both SDL and CC are again at quite high abstraction level, and both backtrack to requirement elicitation and requirement management. SDL also recommends threat modelling. Based on the tool provided[1], Microsoft has a sufficiently lightweight approach to this task. However, modeling a large software system with multiple servers and interfaces, and maintaining that model through the iterations may become a burden; also, the model should be reviewed as any other artifact in order to maintain a credible security tool.

SAMM contains two categories for this phase: the security architecture and a design review. Security architecture consists of a list of practical procedures: first, maintaining a list of recommended software frameworks and applying security principles to the design; second, security services and infrastructure are to be identified an promoted, and security design patterns identified from the architecture. The third level does not include development-time architectural activities; it calls for formal reference architectures and platforms are to be established and frameworks, patterns and platforms validated. Design review, at the first maturity level, should include identification of attack surfaces and design analysis against the security requirements. Requirements for the second maturity level are inspection for complete provision of security mechanisms and the organizational task of deployment of design review service for project teams. Third level is again project specific, containing the activities of developing data-flow diagrams for sensitive resources and establishing release gates for design review.

Agile development support security design well: iterations (A1) allow revisiting the earlier decisions and the iteration backlog (A3) as necessary. Agile practises promote simple design (AP3); all security designing and reviews are performed under this activity. TDD (AP2) and pair programming (AP7) convey the security design into implementation and verification phases. Iterations (A1) implicitly offer opportunities to enhance the security design in case the requirements or environmental factors have changed. Having customer on-site for communication (AP6) also supports security design process.

[1]https://www.microsoft.com/en-us/download/details.aspx?id=49168, ref. 19. Feb. 2018

Table 5: Implementation phase activities

Source	Security activity	Agile activity
SDL	Use Approved Tools	A1
SDL	Deprecate Unsafe Functions	A1, AP1
SDL	Perform Static Analysis	AP7
CC	Analysis and checking of processes and procedures	AP7
SAMM	Implementation review	AP7

3.4 Implementation phase

Table 5 contains the security activities for the implementation phase, necessary to achieve the security objectives. Mainstream software engineering is increasingly dependent on a large set of interconnected and connected tools. Programming IDEs integrate into packet management servers and code repositories; code repositories are part of Continuous Integration and Continuous Delivery (CI/CD) services, and automated unit tests are executed upon each commit. Automated CI/CD systems deploy the tested components into staging areas, from where the code will eventually be released into production after integration and security testing. Although a lot of the security-related implementation-time activity can be automatized, a human-performed static code review is considered very effective. Continuous integration (AP4) is a common agile practise.

Static reviews are the only implementation-time activity in SAMM, and the model gives quite coherent way to conduct them: review checklists are created, and high-risk code is somehow identified and reviewed in detail. At second level automated analysis tools are to be used, and the code analysis to be integrated into the development process. On the third level, the code analysis automation is to be made application-specific and release gates for the review established.

Coding standards (AP1), although established already in the pre-requirement phase, are an important quality improvement practise. It also directly contributes towards security by enabling code reviews and making the source code more structured. Pair programming (AP7) is a very effective quality and security improvement practise [27]; pair programming also acts as a substitute for formal reviews [11]. Iterative development (A1) gives opportunities for refactoring (AP5) which also works as security improvement measure; activities and practises such as daily meetings (A5) and underlying TDD (AP2).

3.5 Verification and validation phase

Security verification activities are presented in Table 6. In order to achieve security objectives and effectively manage security requirements, the iterative security verification faces two unique issues:

(1) Returning of the failed items into the backlog, accounting requirement and design changes.

Table 6: Verification phase activities

Source	Security activity	Agile activity
SDL	Perform Dynamic Analysis	AP4
SDL	Perform Fuzz Testing	A4
SDL	Conduct Attack Surface Review	A4
CC	Verification of proofs	A4
CC	Independent functional testing	AP4
CC	Test case and test result review	A4
CC	Penetration testing	A4
CC	Verification of processes and procedures	A4
SAMM	Security testing	AP4

(2) Automating the security testing, or performing it in such manner that each potentially shippable iteration has gone through the security verification process.

Test-Driven Development (AP2) is an obvious enabler for security verification practises; proper training in security and testing methods should help incorporating the security testing into the software testing suite. Costly and time-consuming fuzz testing, advocated by the SDL, can be considered a quite specialized operation, appropriate for organizations developing APIs and operating systems; application developers should have less use for fuzzing.

The security verification phase is best covered in security engineering methodologies, and has the least direct counterparts in agile activities. Performing these security activities as part of security assurance procurement is to be done already at the requirement specification phase, and the security requirements inserted into the product backlog (A4). Security verification can also be performed during a security specific iteration. Continuous integration (AP4) automates and helps facilitate testing; daily meetings (A5) also often cover testing issues.

SAMM continues to rely on the security requirements also in the verification phase: security test cases are drawn from them. SAMM also calls for penetration testing at the basic maturity level, an activity that requires specific knowledge and tools, and is typically performed by security engineering experts. Only at level two does SAMM require use of automated security testing tools and integration of security testing into development process. Similarly to the implementation verification, on third level, the automation is to be made application-specific and release gates for security testing established.

3.6 Release phase

At the end of each iteration a potentially shippable program increment is released, and the activities in Table 7 are to be performed. The only agile activity taking place at the release phase is the retrospect (A6). This quality improvement measure is directly applicable to security engineering as well. Security engineering activities taking place at later phases of the software lifecycle are crucial to the security objectives, but separate from the development process. Continuous integration (AP4) also extends in the release of the software; on-site customer (AP6) participates also in release-time activities.

In the Common Criteria, security is verified through two processes: security functionality is verified by functional testing, and security assurance by documentation reviews. At the project's inception, one of the security-related objectives is setting of the Evaluation Assurance Level (EAL). This level, ranging from 1 (most basic) to 7 (the most rigorous) defines the amount of documentation to review. The formality requirement for the software code itself increases accordingly. The development-time documentation consists of five parallel documentation tracks, number and level of which is increasing as the EAL rises. At EAL 1, only basic functional specification is required. EAL 2 adds a basic design document, and security architectural description; it also requires the basic functional specification to be augmented with specification of security-enforcing functionality. Each level brings additional documentation requirements up to EAL 5, after which the documentation or formalization requirements do not increase. The maintenance-specific documentation is not included in the development-time documentation requirements.

Table 7: Release-time activities

Source	Security activity
SDL	Create an Incident Response Plan
SDL	Conduct Final Security Review
SDL	Certify Release and Archive
CC	Analysis of guidance documents
SAMM	Issue management
SAMM	Environment hardening
SAMM	Operational enablement

SDL is less concerned with the internal documentation and concerns on things such as certification and maintenance. This is well in accordance to claims that majority of the cost in software engineering incurs after the release phase [16]. This is reflected in SAMM's trio of activity categories directed at security of the post-development part of the software lifecycle: SAMM calls for issue management, environment hardenings and enabling the operations teams for security, before the software can be released. The last of these provides actual tasks for the software development phase: at first level, critical security information should be captured, and procedures (operational instructions) for typical application alerts documented. At second level, change management procedures are created, related to the issue management, and formal operational security guides created and maintained. Third level again concentrates on the business goals, and calls for an audit program and code signing.

DevOps models call for maintenance phase activities as well; for the purposes of this study, the release phase is considered to cover the essentials required for setting up the maintenance operations. A primary concern would be setting up a process to bring the defects found in operations into the development backlog, and giving them a sufficient priority.

4 DISCUSSION

Development-time security activities form the basis for the software security. The security functionality, security assurance and operational documentation are created by the development-time processes, forming the base for the later phases of the software lifecycle. Figure 2 gives an overview of the difficulty in direct mapping

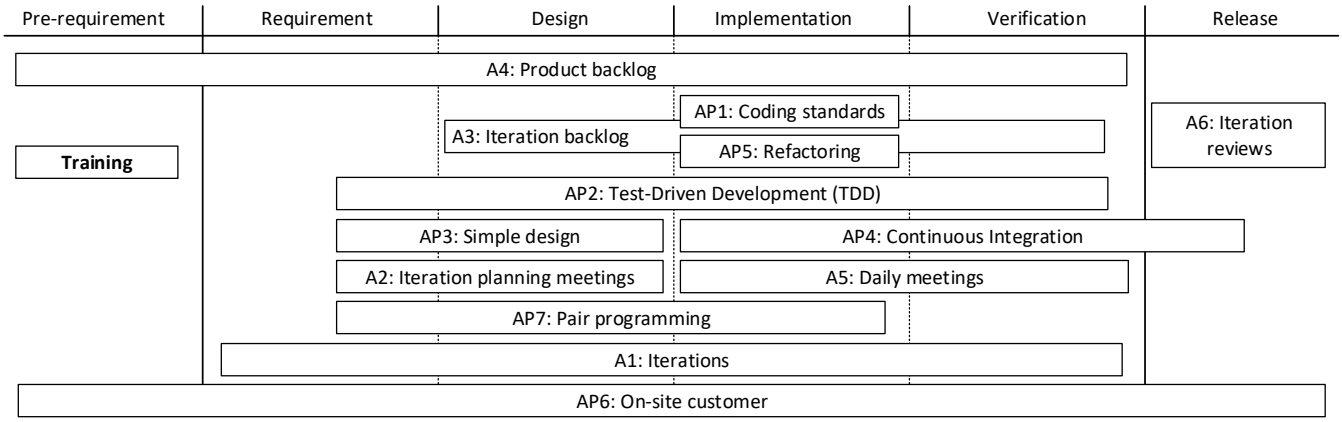

Figure 2: Agile activities mapped into the security development life cycle.

of sequential lifecycle models to agile development practises; very few agile activities are confined into a single lifecycle phase, and even this division appears somewhat artificial in a dynamic agile model.

Pre-requirement tasks are mostly policy and process oriented; security training, training in the agile processes, and establishing practises such as coding standards takes place in this phase – and occasionally these practises may have to be revisited even during the course of the implementation process. In iterative development, the phases from here onward are iterative. The phases are bound together by most important requirement elicitation method: communication with the customer, reflected by "on-site customer". Product backlog is also established already at the pre-requirement phase with the general requirements, which are then complimented with the project specific items at later phases. Test-driven development binds the phases together; pair programming, even if utilized only at crucial points and as an "enhanced code review practise", is another iteration-spanning activity useful from the requirement phase onward, until end of implementation phase.

Simple design, iteration planning, and placing the planned items into an iteration backlog bind the requirements and design together; implementation is augmented with daily communication between the developers, and also availability of the customer communications. Continuous Integration, together with TDD, provides security verification management and proper coordination with iteration backlog by identifying the items that require rework; also refactored items requiring security verification are handled through these activities – automatically, with proper tooling. CI process also produces the releases after the verification, to undergo any release-phase security activity after completion of the development processes.

Security development lifecycle models have a strong emphasis on security verification. This is also the phase with least common activities between the agile and security engineering activities. The solution to this discrepancy is twofold: both strong integration of security testing into the functional testing and CI/CD processes is required; also, a level of pre-planning is required: the security engineering activities required to achieve the security objectives have to be recognized early in the development process and the activities placed into the the product backlog.

Agile methods are geared towards requirement management and getting features implemented and delivered; however, security experts still keep reporting that this is not the case with security objectives [see 36]. While value-driven agile development processes have certain unique shortcomings, fulfilling security requirements could be as simple as a matter of prioritization. As long as the security personnel and security requirements are external, the security objectives are under the threat of getting poorly realized, if at all. This can only be changed by increasing the awareness of the security engineering processes and including the security features and especially verification activities into the development process itself. As long as security engineering is external to the software development, also security objectives remain external – at the cost of potentially inadequate software security.

5 CONCLUSION

Implementing software with security objectives requires alignment of software engineering and security engineering processes by infusing the security engineering activities directly into the agile processes. This should take place on three levels: providing training for the individuals, executing security requirement management, and by integrating the security activities, tools and experts into the software development process. With an acceptable level of preliminary planning the security-related work items are to be placed into the product backlog, and completed at a convenient time during the iterative development process.

Achieving security objectives in software development requires security engineering. Software security is an investment: it requires training, tools and time. Integrating security engineering directly into the software development activities, rather than executing it as detached processes, is intuitively an obvious benefit – both economically and technically. This study has provided a framework for this alignment, and suggested ways to overcome the potential difficulties in this alignment process. Software development is agile, and security engineering will have to follow suit.

REFERENCES

[1] Scott W. Ambler and Mark Lines. 2012. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise* (1st ed.). IBM Press.

[2] Tigist Ayalew, Tigist Kidane, and Bengt Carlsson. 2013. Identification and Evaluation of Security Activities in Agile Projects. In *Secure IT Systems: 18th Nordic Conference, NordSec 2013, Ilulissat, Greenland, October 18-21, 2013, Proceedings*, Hanne Riis Nielson and Dieter Gollmann (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 139–153. https://doi.org/10.1007/978-3-642-41488-6_10

[3] 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. https://doi.org/10.1145/1987875.1987900

[4] Richard L. Baskerville, Lars Mathiassen, and Jan Pries-Heje. 2005. Agility in Fours: IT Diffusion, IT Infrastructures, IT Development, and Business. In *Business Agility and Information Technology Diffusion*, Richard L. Baskerville, Lars Mathiassen, Jan Pries-Heje, and Janice I. DeGross (Eds.). Springer US, Boston, MA, 3–10.

[5] Kent Beck. 2000. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[6] Konstantin Beznosov and Philippe Kruchten. 2004. Towards agile security assurance. In *NSPW '04 Proceedings of the 2004 workshop on New security paradigms*. 47–54.

[7] Barry Boehm. 2006. Some future trends and implications for systems and software engineering processes. *Systems Engineering* 9, 1 (2006), 1–19. https://doi.org/10.1002/sys.20044

[8] B. Boehm and R. Turner. 2003. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, New York.

[9] B. Boehm and R. Turner. 2003. Using risk to balance agile and plan-driven methods. *Computer* 36, 6 (June 2003), 57–66. https://doi.org/10.1109/MC.2003.1204376

[10] B. Boehm and R. Turner. 2005. Management challenges to implementing agile processes in traditional development organizations. *IEEE Software* 22, 5 (Sept 2005), 30–39. https://doi.org/10.1109/MS.2005.129

[11] Alistair Cockburn and Laurie Williams. 2000. The costs and benefits of pair programming. *Extreme programming examined* 8 (2000), 223–247.

[12] Common Criteria Recognition Arrangement (CCRA). 2018. The Common Criteria. (2018).

[13] Jessica Diaz, Juan Garbajosa, and Jose A. Calvo-Manzano. 2009. Mapping CMMI Level 2 to Scrum Practices: An Experience Report. In *Software Process Improvement*, Rory V. O'Connor, Nathan Baddoo, Juan Cuadrago Gallego, Ricardo Rejas Muslera, Kari Smolander, and Richard Messnarz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 93–104.

[14] Edsger W. Dijkstra. 1982. *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag.

[15] Brian Fitzgerald and Klaas-Jan Stol. 2014. Continuous Software Engineering and Beyond: Trends and Challenges. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering (RCoSE 2014)*. ACM, New York, NY, USA, 1–9. https://doi.org/10.1145/2593812.2593813

[16] R. L. Glass. 2001. Frequently forgotten fundamental facts about software engineering. *IEEE Software* 18, 3 (May 2001), 112–111. https://doi.org/10.1109/MS.2001.922739

[17] Johannes Holvitie, Sherlock A. Licorish, Rodrigo O. Spínola, Sami Hyrynsalmi, Stephen G. MacDonell, Thiago S. Mendes, Jim Buchan, and Ville Leppänen. 2017. Technical debt and agile software development practices and processes: An industry practitioner survey. *Information and Software Technology* (2017). https://doi.org/10.1016/j.infsof.2017.11.015

[18] Michael Howard and Steve Lipner. 2006. *The Security Development Lifecycle*. Microsoft Press, Redmond, WA, USA.

[19] ISO/IEC standard 15408-1:2009. 2009. *Information technology – Security techniques – Evaluation criteria for IT security*. ISO/IEC.

[20] ISO/IEC standard 21827. 2008. *Information Technology – Security Techniques – Systems Security Engineering – Capability Maturity Model (SSE-CMM)*. ISO/IEC.

[21] S. A. Licorish, J. Holvitie, S. Hyrynsalmi, V. Leppänen, R. O. Spínola, T. S. Mendes, S. G. MacDonell, and J. Buchan. 2016. Adoption and Suitability of Software Development Methods and Practices. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. 369–372. https://doi.org/10.1109/APSEC.2016.062

[22] CMMI Institute LLC. 2017. The CMMI Institute. (2017). http://cmmiinstitute.com/.

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

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

[25] OWASP. 2018. OWASP Top 10 Application Security Risks. (2018).

[26] 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. https://doi.org/10.1109/ARES.2016.103

[27] M. C. Paulk. 2001. Extreme programming from a CMM perspective. *IEEE Software* 18, 6 (Nov 2001), 19–26. https://doi.org/10.1109/52.965798

[28] Balasubramaniam Ramesh, Lan Cao, and Richard Baskerville. 2010. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal* 20, 5 (2010), 449–480. https://doi.org/10.1111/j.1365-2575.2007.00259.x

[29] Kalle Rindell, Sami Hyrynsalmi, and Ville Leppänen. 2017. Busting a Myth: Review of Agile Security Engineering Methods. In *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES '17)*. ACM, New York, NY, USA, Article 74, 10 pages. https://doi.org/10.1145/3098954.3103170

[30] P. Rodríguez, J. Markkula, M. Oivo, and K. Turula. 2012. Survey on agile and lean usage in Finnish software industry. In *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. 139–148. https://doi.org/10.1145/2372251.2372275

[31] Reijo M. Savola, Christian Frühwirth, and Ari Pietikäinen. 2012. Risk-Driven Security Metrics in Agile Software Development - An Industrial Pilot Study. *j-jucs* 18, 12 (jun 2012), 1679–1702. |http://www.jucs.org/jucs_18_12/risk_driven_security_metrics|.

[32] Ken Schwaber. 1995. Scrum Development Process. OOPSLA'95 Workshop on Business Object Design and Implementation.

[33] S. Subashini and V. Kavitha. 2011. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications* 34, 1 (2011), 1 – 11. https://doi.org/10.1016/j.jnca.2010.07.006

[34] Synopsys Software Integrity Group. 2017. The Building Security In Maturity Model. (2017). https://www.bsimm.com/

[35] I. A. Tondel, M. G. Jaatun, and P. H. Meland. 2008. Security Requirements for the Rest of Us: A Survey. *IEEE Software* 25, 1 (Jan 2008), 20–27. https://doi.org/10.1109/MS.2008.19

[36] Sven Türpe and Andreas Poller. 2017. Managing Security Work in Scrum: Tensions and Challenges. In *Proceedings of the International Workshop on Secure Software Engineering in DevOps and Agile Development (SecSE 2017)*. CEUR Workshop Proceedings, 34–49.

[37] VersionOne. 2017. 11th Annual State of Agile Survey. (2017). https://versionone.com/pdf/VersionOne-11th-Annual-State-of-Agile-Report.pdf.

[38] John Viega and Gary R McGraw. 2002. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley.

[39] William Wake. 2003. INVEST in Good Stories, and SMART Tasks. (2003). https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/