Security in Agile Software Development: A Practitioner Survey

Kalle Rindell[a], Jukka Ruohonen[a], Johannes Holvitie[a], Sami Hyrynsalmi[b,c], Ville Leppänen[a]

[a]*Department of Future Technologies, University of Turku, FI-20014 Turun yliopisto, Finland*
[b]*Unit of Computing Sciences, Tampere University, P.O. Box 300, FI-28101 Pori, Finland*
[c]*Software Engineering, Lappeenranta-Lahti University of Technology LUT, Mukkulankatu 19, FI-15210 Lahti, Finland*

Abstract

Context: Software security engineering provides the means to define, implement and verify security in software products. Software security engineering is performed by following a software security development life cycle model or a security capability maturity model. However, agile software development methods and processes, dominant in the software industry, are viewed to be in conflict with these security practices and the security requirements.
Objective: Empirically verify the use and impact of software security engineering activities in the context of agile software development, as practiced by software developer professionals.
Method: A survey (N=61) was performed among software practitioners in Finland regarding their use of 40 common security engineering practices and their perceived security impact, in conjunction with the use of 16 agile software development items and activities.
Results: The use of agile items and activities had a measurable effect on the selection of security engineering practices. Perceived impact of the security practices was lower than the rate of use would imply: This was taken to indicate a selection bias, caused by e.g. developers' awareness of only certain security engineering practices, or by difficulties in applying the security engineering practices into an iterative software development workflow. Security practices deemed to have most impact were proactive and took place in the early phases of software development.
Conclusion: Systematic use of agile practices conformed, and was observed to take place in conjunction with the use of security practices. Security activities were most common in the requirement and implementation phases. In general, the activities taking place early in the life cycle were also considered most impactful. A discrepancy between the level of use and the perceived security impact of many security activities was observed. This prompts research and methodological development for better integration of security engineering activities into software development processes, methods, and tools.

Keywords: survey, security engineering, agile software development, software security, security standards, security assurance

1. Introduction

Secure software development is performed by executing a set of security engineering activities in conjunction with software development processes [1, 42, 78]. Purportedly this is done by following a security development life cycle model, or an implementation of a security maturity model. However, in agile software development the execution and progress of software development processes cannot always be determined in advance – nor are they necessarily even predictable at the start of a project.

This premise complicates the software development processes in the form of production bottlenecks. These bottlenecks include the production of additional documentation, performing security-related reviews and scans, and time-consuming security testing. Strict process requirements constrain the flexibility gained through the use of agile methods, and goes against the principles of adaptable processes [59, 67, 75]. The agile techniques are applied to serve specific purposes, as do the security engineering techniques. When both elements are present in a software project, security must not be compromised by the strive for agile principles and lightweight processes; conversely, security processes should not decrease the effectiveness of the development process.

Security incidents and threats are a significant factor when determining the cost of creating a software product and maintaining it throughout its lifetime [46]. Secure software engineering involves clearly defined security objectives, elicitation of requirements, creation of security architecture and design, and secure programming practices. Security assurance is gathered throughout the life cycle, and by verifying the implemented security features. Regulative requirements and rigorous security frameworks typically introduce additional security assurance requirements. These take the form of additional documentation, security reviews, security testing, and audits [29, 69]. A main objective of software security engineering is to provide the software developers means to comply with security assurance requirements. The general aim of software security engineering is to address the identified software security risks already in the development phase, enabling efficient creation of effective security solutions.

Introducing new tasks and tools into software development workflow necessarily incurs a cost in time and other resources. Software development organizations operate under strict budgeting and scheduling restraints, with careful monitoring of the ability to produce deployable software. These restraints have created a high demand for efficient software development processes, including those specific to security engineering. This stresses the importance of efficiently combining the agile software development and security engineering processes, and the significance of this task to both industry and research.

To determine the state of the art practices in software security, an industry survey was performed among the Finnish software industry practitioners. The respondents were asked about individual agile software development and software security engineering activities they use during software development, and the security impact of the security practices used. In total, the survey contained questions about the usage of forty common software security activities and twelve agile practices, along with their perceived impact on software security.

The rest of this article is structured as follows: The opening Section 2 provides the theoretical background and motivation for the survey. The survey design is described in Section 3, and the results are presented in Section 4. Results and implications are further discussed in Section 5; examination of threats to validity conclude the article in sections 6 and 7 respectively.

2. Background

This section provides an introduction to the main concepts and challenges software security engineering is facing, and the regulatory and industrial background of security in software development.

2.1. Software security engineering

The purpose of software security engineering is to identify, mitigate, and avoid security threats to software and data assets [1, 42]. In this approach, the goal of ensuring the availability, integrity, and confidentiality of information is achieved by security policies, implemented into software security features during software development. To verify the implementation, appropriate security assurance is produced and gathered along the development, testing and operations.

As a means to increase software dependability, software security engineering aims to guarantee the correctness of software, computer networks, and computer systems in adverse situations. The problem field in software security is relatively well-known [73], whereas the common approach to mitigate and manage security is often performed by following relatively simple checklists to guide design and implementation practices [27, 51, 62]. However, software security engineering must provide also more systematic means to address the security risks in software development. Software dependability, a related concept, is achieved through such means as fault prevention, fault tolerance, fault removal, and fault forecasting [2].

Emphasis on the prevention of latent security issues necessarily places software security engineering to the early phases of the software life cycle. This provides an opportunity to prevent security incidents from occurring in the first place. It can also supplement the external security mechanisms introduced in the operations and maintenance phases of the applications' life cycle. A common characteristic of security faults, such as implementation bugs and design flaws, is their persistence. When not repaired, they become "features", requiring constant and costly security engineering activities throughout the software's operational lifetime. Eliminating such faults early on requires significantly less resources than having to fix the issues later [35, 55]. Furthermore, hardware errors, problems caused by users, and other issues in the operating environment are more transient and can be mitigated by means independent of the assets protected [26]. These characteristics make software security engineering distinct from other security domains.

Systematic security assurance contains rigorous documentation of technical security features, security architecture, and procedural security instructions. A number of security verification techniques may also be used [69]. These techniques—various forms of security documentation, testing, reviews, and audits—seek to detect flaws and errors, and to provide an appropriate level of assurance.

2.2. Software security development models

Software security engineering consists of a diverse set of techniques and processes. In research, they are often arranged into security development and life cycle models. Organizational security practices are traditionally presented in the form of security maturity models, promoting repeatability and continuous improvement of the security activities used in software development.

To make the investment in software security more feasible for the software development organizations, also less demanding maturity models have been developed. An example is the openly available Software Assurance Maturity Model (SAMM) [52]. SAMM makes an explicit claim of being "agile agnostic": In practice, this means that it does not even mention the principles and values of the Agile Manifesto [7]. It does, however, take into consideration the tremendous advancements in continuous and automated software engineering practices and processes. SAMM is also accompanied with a diverse set of open source tools and frameworks, including those provided by the Open Web Application Security Project (OWASP). These less demanding yet still rigorous maturity models encourage developers and organizations to tailor the models to their own processes and to perform at least the most necessary security improvements. This tailoring aligns with various industry best practices, often published in the form of security checklists and "do not do" lists [27, 51, 62]. Like with maturity models for agile development [50], it should be noted that these security-specific maturity models and their checklists are not adequate in all contexts.

The SSDLC models used were the Microsoft Security Development Lifecycle (SDL) model [78], and Touchpoints for Software Security [42]. The latter also forms the development life cycle model included in the Building Security In Maturity Model (BSIMM). To comply with the changing software development practices, Microsoft later made a rudimentary effort to adopt the SDL for agile development [25]. However, the references to agile development have since been removed from the current version of the SDL [43], and the life cycle model itself integrated into DevOps processes and tools used in Microsoft's public cloud service.

The international standard for security development framework has been created by the ISO/IEC in the form of Secure Software Engineering Capability Maturity Model (SSE-CMM) [31]. This model claims to contain the best practices for security engineering. It formalizes security work into an exhaustive set of security processes, by defining 129 security processes divided into 22 process areas. To supplement this high-level standard, multiple international, national and domain-specific security regulations have been crated [22, 55].

In Finland, where this survey was performed, a comprehensive set of information security instructions has been issued by the Government Information Security Management Board (Finnish abbreviation: VAHTI [76]). This model contains a specific Software Security Development Life Cycle (SSDLC) model. While this model does not cite any references, it is largely formed after the SDL and Touchpoints, and based on the Common Criteria [30], the ISO standard for software product evaluation, and appears to conform with the ISO application security standard [28]; VAHTI can be considered to represent the state of the art of software security engineering practices in Finland. KATAKRI is another Finnish security framework, geared towards compatibility with European and North American security regulations.

The most notable industrial software security framework outside the SSDLCs and maturity models is SAFECode [58]. This approach to security management is based on a rigorous security risk management process. Similarly to ISO standard framework, SAFECode addresses the identified risks by secure design and coding practices coupled with strict management of third party software components. A security incident response function is used to cover the operational part of a software life cycle. SAFECode supplements the security maturity models by providing concrete and actionable instructions for the deployment of security processes. It also contains the essential principles guiding the build-up of an SSDLC-enhanced software development process.

2.3. Related work

Applying security engineering methods and models into agile software development was initially seen as challenging [15, 79]. As agile development matured experiences have often been more positive, although even recent empirical research continues to report organizational and technical problems in adaptation [75]. The perceived mismatch between formal security engineering and software development has prompted many attempts to combine maturity models with agile processes [65, 67]. A finding common to these studies is that agile software development can comply even with strict and formal security requirements, but at the unsurprising expense of slower development and higher cost, largely due to non-agile security processes.

In software development, agile methodologies promote and provide the ideals of flexibility and freedom [6, 64]. Even though the agile methods were initially criticized due to their perceived contradictions with the traditional process-oriented approaches [74, 48], agile principles have since been successfully adopted also to the regulated areas of the software industry. Agile methods are now used to fulfill extensive software safety and security requirements, although this may require compromising some of the benefits of the agile approach. The iterative methods also contain inherent mechanisms for continual performance and quality improvement. Agile principles simplify that tradition, and build upon it [66].

Iterative and incremental software development methods have inspired and enabled new continuous integration and deployment models, such as DevOps, which considerably shorten maintenance cycles. In terms of security, this shortening manifests itself through accelerated delivery of security improvements and quicker recovery from security incidents. However, in regulated environments, DevOps has also definite challenges caused by the introduction of new practices [36]. Despite of these challenges, a growing number of organizations is utilizing automated processes to deploy software into production [8]. The tools, techniques, and methodologies used for the automation contain also security considerations. To support continuous delivery, further work has been done to find an acceptable level of continuous software security using agile methods [10].

Given the challenges outlined, it is not surprising that previous work has been done to examine software security engineering in the context of agile software development. Early contributions for agile software security engineering were mainly theoretical concepts on how to perform and manage agile software security engineering. Given that software architectures have often been seen as a weak point of agile methods [9, 66], a good example would be the suggestions for secure software architectures in agile development [13, 18]. Similarly, suggestions for producing security assurance using Extreme Programming (XP) have been made [11]. This line of work has also provided early outlines for creating and gathering assurance at requirements, design, implementation, and testing phases.

Elicitation of security requirements, whether through misuse cases or by other means, has also been studied [34, 72].

A major research challenge centers around the question of how to integrate security engineering practices, maturity models, and activities required to comply with standards into flexible agile work flow [56]. The main objective should not be maintaining "agility" as such, but to produce as secure software as necessary, as efficiently as possible. The many quality-improving mechanisms in agile development—iterative development, retrospectives, constant refactoring, and continuous integration—work towards both of these goals when producing secure software [5].

The same applies in the safety context [32]. Also the software security life cycles in an agile context have been a frequent subject in software security research [3, 4, 12]. General challenges in adapting security engineering into agile development have been identified [20]. A further example would be the integration of security engineering and education [53, 81]. However, there is a notable lack of industry surveys directly concentrating on the actual development-time activities.

Directly comparable industry surveys are limited both in numbers and in scope. Apart from a small web survey (N = 46) concerning the general use of security life cycle models [19], no surveys about industry use of security engineering practices in software development are known. The survey reported in the present paper fills this notable gap in previous research.

3. Research design

This chapter summarizes the research objective and the research questions. it also describes the practical measures taken to answer those questions, and how to replicate the research process. In the design and implementation of the research, the general good practices for industry survey were followed [68].

3.1. Research questions

The primary research objective was to improve the understanding of the use of security engineering activities in the context of agile software development. The first research question (RQ) makes the research objective explicit:

RQ1: To what extent are security engineering activities utilized in the context of agile software development?

This research question was addressed by querying the extent of usage of both security engineering and agile activities in software projects. Forty software development security practices were extracted from various security engineering models: the ISO/IEC Common Criteria, Microsoft SDL, BSIMM, and VAHTI; three additional activities were extracted from literature [59]. These practices were assembled into five groups according to its phase in the SSDLC: requirement, design, implementation, verification, and release.

The secondary objective augments RQ1 with the following question:

RQ2: What is the perceived security impact of security engineering activities in agile software development?

The two research questions complement each other; the answer to this question was found by directly asking for the perceived impact of each used activity. The use of a particular security engineering activity may or may not reflect its perceived impact. The activity may be used because it is efficient for improving security. However, it may be used also because it demands only few resources or because it is easy to embed into existing agile software development work practices. The reverse relation is also important: the activity may be used only infrequently even though its perceived impact upon security is substantial. Answers to RQ1 and RQ2 can implicitly pinpoint important insights about potential incompatibility problems between agile methods and security engineering activities.

3.2. Questionnaire

The survey was explicitly designed to find answers to research questions RQ1 and RQ2. To ensure a representative sample, the population targeted refers to Finnish software engineers, software security specialists, and other direct participants in software development processes. For evaluating the success of this targeting, relevant background information was queried from the respondents. The information queried includes a few structured questions on education, work experience, software development role, security training acquired, and organizational aspects (including company size, application area, and potential regulations or certifications for projects under delivery). As both research questions concentrated on agile software development, the questionnaire included also a group of questions designed to measure the agility of the projects. To achieve this, the questionnaire included a structured question about the agile software development methodologies used in the projects the respondents use. The most common methodologies at the time of the survey, Extreme Programming (XP) and Scrum, were used as specific examples.

To measure how 'agility' was achieved, the use of sixteen work practices and processes typical to agile software development were presented in the questionnaire. Many of these are derived directly from the classical Agile Manifesto [7]. More importantly, the sixteen selected agile work practices were the same as the ones used in a recent study on technical debt [24], sans retrospectives that were omitted. This can be used as a data point in any follow-up studies for longitudinal longitudinal. The examples include test-driven development, refactoring, continuous integration, iteration backlog, and rest of the usual suspects.

To avoid clustering the activities into verification phase, review activities were principally placed into the lifecycle phase where the item or artifact is created. Code-related reviews are thus placed into implementation phase, as were requirement and design reviews placed into their respective phases. This was done to partially reflect the phase into which the activity has most impact, and partially to keep the amount of options in the questionnaire manageable for each lifecycle phase.

A pilot group of twenty persons was used to pre-test the questionnaire. Based on the feedback, a few corrections were made to the wording of the questions and the terminology. Given that not all of the associated terms are unambiguously defined even in scholarly research [71], particular attention was paid to the descriptive names of the security engineering activities. These were formulated to conform with the Microsoft SDL, VAHTI, and the BSIMM surveys, with specific attention paid to the BSIMM terminology to gain commensurability. A secondary naming principle was to use the VAHTI names for the activities in cases where the sources differed.

3.3. Scales

The agile and security engineering practices were surveyed using the following five-point Likert scale:

5. Systematically used throughout the projects
4. Mostly used throughout the projects
3. Sometimes used during the projects
2. Rarely used during the projects
1. Never used during the projects

This scale was used based on the premise that development processes are not fixed in agile software development. Therefore, the value five was phrased with the word "systematically" rather than with the word "always". Otherwise the scale was comparable to the ones used in existing surveys (notably, [70]). The same scale was used in the questionnaire for all security engineering activities. An analogous five-point scale was used for the perceived impact of these:

5. Very high
4. High
3. Moderate
2. Low
1. Very low

The online answering about the perceived impact of the activities was streamlined by dynamically hiding the activities that were unused.

3.4. Survey implementation

The research was conducted as an invitation-based online survey. There were two types of invitations: a public web link and targeted links sent by personal e-mail invitations. The individual invitees were collected with two distinct methods from a precompiled list of software companies.

The survey was announced and first invitations sent by a public invitation via a monthly news letter of the Finnish Software Entrepreneurs Association. This invitation contained the public web link to the survey. The coverage of the news letter was about 340 executives in the Finnish software companies. The recipients were also asked to further share the link to their employees. This method, however, was considered to be inadequate for reaching the targeted audience of software engineers and software security specialists. Furthermore, the invitation was sent during the holiday season, necessitating the second method for invitations.

The second phase of invitations involved two methods. The first was manual compilation of invitee email addresses from the websites of 303 Finnish software companies. In the second method, 69 software companies were targeted by searching the corresponding employees from LinkedIn. Following an existing research example [44], LinkedIn's "People Search" functionality was utilized for email address gathering; LinkedIn was selected for its specific focus on professionals.

For ethical reasons and to respect employee privacy only companies with an expressly published email addresses or address forming policies were searched. The second method produced the best targeted invitees, but has weaknesses in e-mail address accuracy as they were formed based on the announced address forming policies.

In the selection process, employees working in non-technical roles were systematically excluded from the population. The exclusion criteria specifically targeted people with functional titles related to finance, human relations, sales, customer support, as well as non-technical executives. Inclusion criteria contained all roles related to software engineering and information security, or technical management of projects, departments or companies, up to C-level.

The survey remained open for a period of six weeks from December 2017 to January 2018. A reminder message was sent after three weeks. In total, 62 valid responses were received. The exact response rate cannot be calculated due to the public invitations used in the first stage, but given that 1,436 email addresses were contacted in the second stage, the response rate can be interpreted to be low, as is typical to online surveys.

A central purpose of the survey was also to elicit industry best practices and expert opinions in software security engineering. This objective was approached through careful selection of the participants, and a very focused selection of activities included in the questionnaire. As a result, a representative and focused sample of population was invited to report on specific types of activities in a well-scoped context. Due to the deterministic reduction of randomness in both the population and the observed phenomena (development time activities), the central limit theorem was deemed not not to apply. Furthermore, the survey revealed a high incidence of self-designed security rules supplementing the regulation-based ones, rendering the independence of the security activities under doubt. In line with our purpose of inspecting these specific phenomena, and presenting relevant and truthful results, the research objective is accomplished using descriptive statistics.

4. Results

This section reports the results as designed: background details of the respondents and their organization, and the answers to the research questions.

4.1. Background

The background questions were used to profile the respondents and their role in the software development. This section also covers basic information about their organizations, certifications, and the application area in which security has been a concern in a software project.

4.1.1. Demographics: experience, education, and organizations

The respondents are generally well-educated and highly experienced in software engineering. While about five percent reported no experience in software development, as much as 77% reported six or more years. Roughly about a half of the respondents have a master's degree, a little below one third have a bachelor's degree, and about fifteen percent have a doctoral degree.

The respondents work in diverse organizations. About 39% worked in organizations with less than fifty employees, about 21% in organizations with 50 to 250 employees, and about 35% in organizations with more than 250 employees. About five percent preferred not to disclose this information. This range presumably reflects the current structure of the Finnish software industry. The same applies to the type of software produced. Most (69%) of the respondents are developing web and cloud applications. The rest are mostly working with desktop and client-server applications, embedded software, and mobile applications. A few specialized domains are also represented, including video games, smart cards, and consulting.

4.1.2. Agile development

The primary targets of the survey were full-time employees involved in technical software development roles. The results of this targeting are shown in Table 1: a total of 87% of the respondents had directly worked in a software development project with security considerations which was managed using an agile methodology. The relevance of evaluation of the research questions is based on this number, also taking into account the relatively high educational levels and long work experiences of the respondents.

Table 1: Project roles

Role	Share (%)
Developer	40
Architect	21
Scrum master or team leader	11
Project manager	10
Executive	8
No projects	5
Security specialist	3
Product owner	2
Tester	0

Table 2: Development methodologies

Methodology	Share (%)
Scrum	30
Scrum-Kanban	28
Custom agile	17
Kanban	14
Other	3
Do not know	5
No answer	3

The respondents were also asked about the particular agile methodologies used in the security-constrained projects they work with. The predefined list of methodologies provided to the respondents was assembled based on a recent industry survey [77]. The results summarized in Table 2 show no surprises: Scrum and its variants are currently the most popular agile methodologies among the respondents. In contrast to earlier surveys [38, 77], none of the respondents reported using XP or a method derived from XP. A few respondents reported not using any specific agile methodology, or declined from naming it. Notably, all of these respondents reported using e.g. iterations and backlogs, both key practices in agile methodologies.

Observed details about the use of agile methodologies are significant on their own rights. Even in security sensitive projects, a majority of respondents managed their work by agile methodologies, and performed agile activities in software development. This observation is particularly significant for the survey at hand: there exists no widely used polar opposite (i.e. use of RAD, 'waterfall', or other non-agile method) for empirical comparisons. The results also justify the wording used for RQ1 and RQ2. Based on the results, it appears that the respondents had performed quite well in adapting agile software development methods for security engineering.

While nearly all of the respondents were working with agile projects, there is still some variance in terms of the actual agile work practices. To query such practices, the respondents were asked to evaluate the use of sixteen different agile work practices on a similar five-point Likert scale used for the security activities (see Subsection 3.3). The results are visualized in Fig. 1, which shows the relative share of the answers across the Likert scale; the

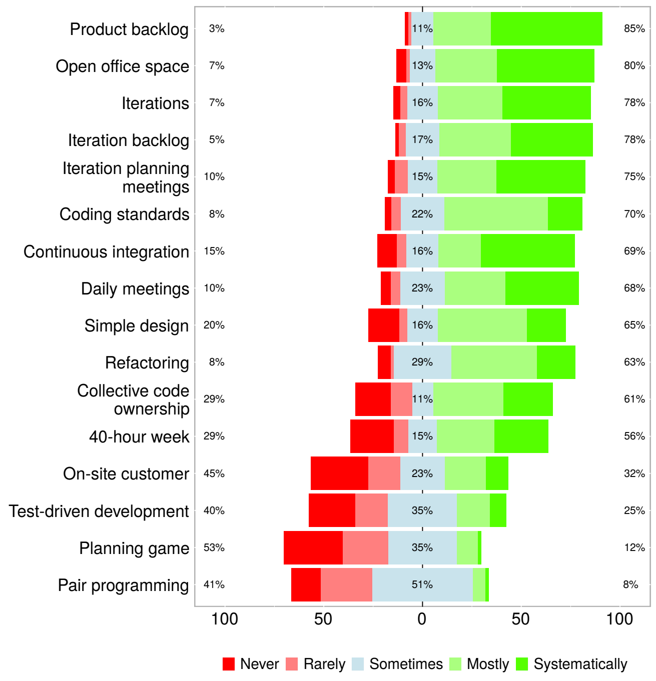

Figure 1: Use of agile work practices

numbers on left (never and rarely) and right (mostly and systematically) of the plot display the combined share of two categories each. When compared to an earlier survey [38] performed in the context of technical debt activities in agile software development, the adoption rates of agile activities by security-oriented developers were consistently higher. Inherently agile activities were prominently used. The examples include product and iteration backlogs, refactoring, iterations, iteration planning, and continuous integration. More generic ones—such as coding standards and open office space—were also very popular. On the side of less frequently used practices were pair programming, planning games, on-site customer, and test-driven development. Even these received a fair amount of use, with only planning game receiving less than half (47%) usage reported as "sometimes" or more.

4.1.3. Standards, regulations, and constraints

A fundamental pillar of agile software development is the explicit involvement of customers throughout the software development processes. This manifests itself also in the typical security constraints imposed upon the software projects surveyed. The data displayed in Table 3 shows that a majority of the projects have constraints that were context-specific and related to customers' particular security requirements. In comparison, the security constraints defined by the software development organizations themselves were used 29% of the projects, with an equal 29% with customer-defined security constraints. The use of various international or national security standards summed up 30%, with 10% using more informal RFC 2196 or other guidelines. Notably, only 2% stated not using any security guidelines.

Table 3: Standards, regulations, and security constraints

Standard, regulation, or constraint	Share (%)
Customer's self-designed constraints	29
Self-designed constraints	29
VAHTI	10
ISO 27000 series	9
KATAKRI	7
ISO 15408 (Common Criteria)	4
RFC 2196 (Site Security Handbook)	4
Other	6
None	2

These observations are significant as regulation is generally regarded as a significant source of security requirements in software development. A fundamental problem with software security standards, such as the Common Criteria, is the omission of business processes that should arguably be the main drivers for security engineering [1]. In this sense, agile practices can patch the limitations of security standards and checklist-style guidelines. It is also important to emphasize that no single standard or framework was dominant among the respondents.

The use of self-designed security constraints typically occurred with use of some formal standard. Use of the national or international security standards typically overlapped with self-designed security constraints. A total of 29% did not report any organizational use of security standards or guidelines. This observation is particularly noteworthy because much of the existing research has focused on international standards and guidelines used by large multinational companies; it is also foreseeable that the already pronounced growing trend in the amount national and international information security frameworks and regulation will continue.

4.2. Use and impact

In the following subsections, the use of each software security engineering activity in each phase is reported. Use of the activities is accompanied by their perceived impact.

4.2.1. Requirements phase

Security requirements are essential to security engineering. Successful requirement management practices may determine the success of the software project. The use of security engineering activities related to requirements is also among the highest in this survey: This observation is visualized in Fig. 2.

The most used security engineering techniques during this phase are the fundamental ones: eliciting the security requirements, defining the goal and criticality, and performing business impact and risk analyses. These activities produce results that are tightly linked with the economic value of the software being produced. However,

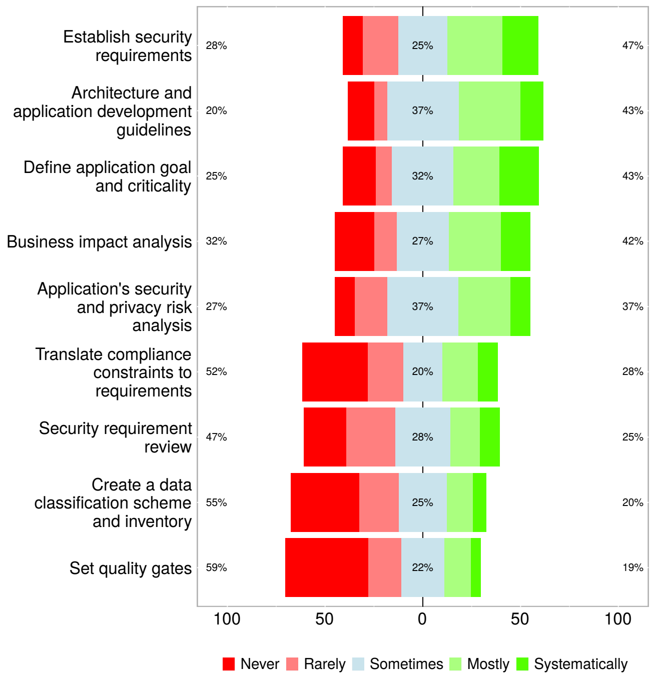

Figure 2: Use at requirements phase

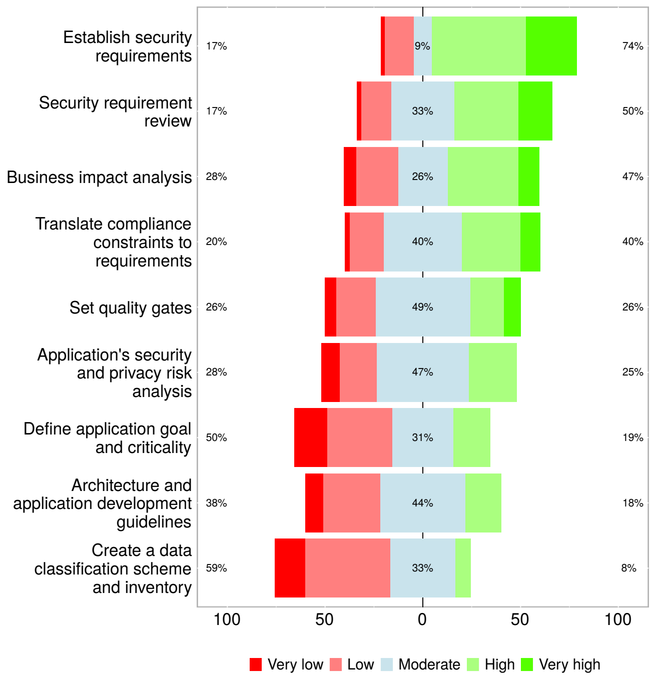

Figure 3: Perceived impact at requirements phase

as can be seen from the analogous Fig. 3, the perceived security impact of some of the activities is less clear. For instance, about half of the respondents perceived the impact of defining an application's goal and criticality as having a low or very low impact upon security. This observation is in a stark contrasts with security requirements, which are often elicited and also perceived as important. Security requirements align also well with the general wisdom about the importance of the early software development phases in security engineering [35, 55]. Interestingly, however, security requirement reviews are a much lesser used activity despite perceived as highly effective. Interestingly, architectural guidelines appear to display an opposite trend, possibly implicating flaws in security architectures, or a common lack of security features in software architectures.

Data classification schemes and creation of quality gates are the least used; these activities were never used by 35% and 42% of the respondents respectively. The perceived impact of creating data classification schemes is also very low. These observations differ from the BSIMM's annual surveys: For instance, creating data classification schemes is one of BSIMM's twelve core activities which, according to BSIMM, everybody does [70]. This is likely due to a shift in techniques: on a provided free-form text field some respondents noted that data classification schemes are typically based on the use cases, and thus do not exist before the software design. Although use cases may be problematic in the security and safety contexts [23], these are still a highly typical characteristic of agile software development. Thus preferring use cases for sensitive data instead of elaborate data classification schemes is a likely explanation for the diverging results.

The prevalence of agile development may also explain the limited use of quality gates—an activity suggested by the Microsoft SDL. These gates can be disruptive and hard to manage in iterative software development. Additionally they constrain the freedom of execution and flexibility of processes, so important to the agile development principles. The lower usage may has a potential explanation may relate to the relatively infrequent use of formal standards and regulations (see Table 3). In general, compliance constraints, typically introducing non-negotiable additional documentation and other assurance activities, were not a particularly common security requirement among the surveyed practitioners: The security requirements were most commonly set by either the customer, or the development organization itself. This result again differs from the BSIMM's surveys [81]. It is also likely for some regulation-based requirements to be included in these customized security instructions without full compliance being achieved.

4.2.2. Design phase

The design phase refers to the work required to translate requirements and abstract architecture principles into more concrete plans for software features and functionality. The technical context of the software produced is clarified and aligned with the results from the requirements elicitation phase. From security point of view, this alignment involves updating of the risk analysis with technical risk definitions and mitigation plans. The security architecture is typically also detailed during this phase.

By implication, the design phase is often seen as particularly crucial for security engineering [22, 55]. The key 'traditional' security engineering tasks in this phase are threat modeling and attack surface recognition. When agile software development is used, many of these activities are carried out using security stories and different misuse or abuse cases [40, 41, 63]. Like in conventional agile development, these activities are used to create security-related tasks to a project's iteration backlog.

Given this general background, the use of the design phase activities surveyed are depicted in Fig. 4. The corresponding Fig. 5 shows their perceived impact.

Application security configurations, design requirements, and abuse or misuse cases are the most frequently used security design activities. These are all very similar to activities performed in most agile software development projects in general: the high usage is likely explained by developers finding it quite natural to add a security flavor into these common activities. The security impact of design requirements and as abuse or misuse cases is also considered high, adding to their perceived usefulness. The security impact of the other design-phase activities is perceived to be significantly lower, making these two activities distinct.

In contrast to the common software design activities, threat modeling and attack surface analysis are exclusively security engineering tasks. This is a likely explanation for their relatively infrequent use. For instance, almost one third of the respondents had never used threat modeling, despite it is promoted by practically all security development models.

There may be also a certain causal logic behind the results: For example, when threat modeling is not done, attack surfaces are not known either. A further examination also reveals that threat modeling was seldom used by those respondents who rarely defined security configurations. These activities are linked to technical analysis of the software, and not necessarily considered contributing to development of new features. While correlation does not indicate causality, low perceived security impact may be a contributing factor to the lack of use, or simply rejection of tasks considered overburdening or difficult to accomplish. Maintenance of a detailed treat model can be an arduous task in iterative development.

4.2.3. Implementation phase

Contemporary software development is typically highly automated. Various tools are used to develop, debug, test, and commit the produced software code and configurations into a repository. Many of these implementation tasks are also directly integrated into development tools. The tools promoted by security standardization bodies [49] include various effective means to improve software quality: most importantly, static analysis tools [14, 80]. The security engineering tasks during implementation reflect these activities and tools. In addition, the security tasks in the implementation consist of security documentation, coding standards, and many related configuration activities supplemented by respective implementation reviews.

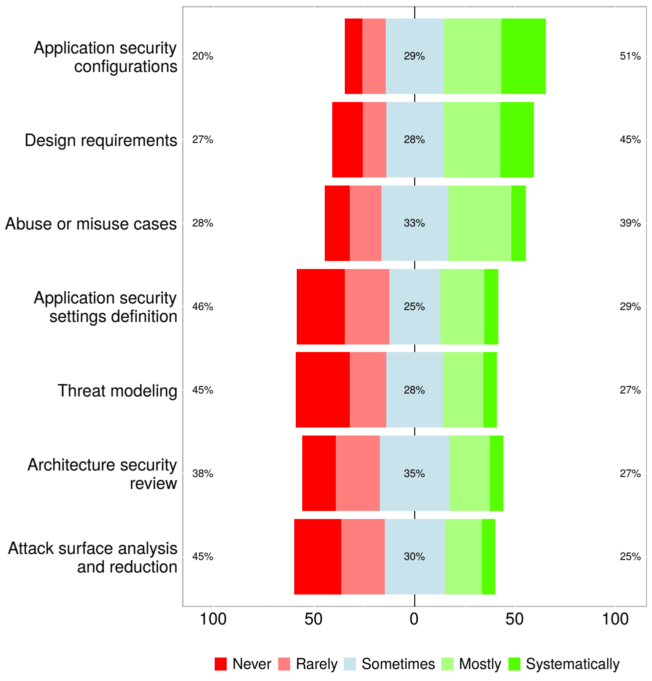

Figure 4: Use at design phase

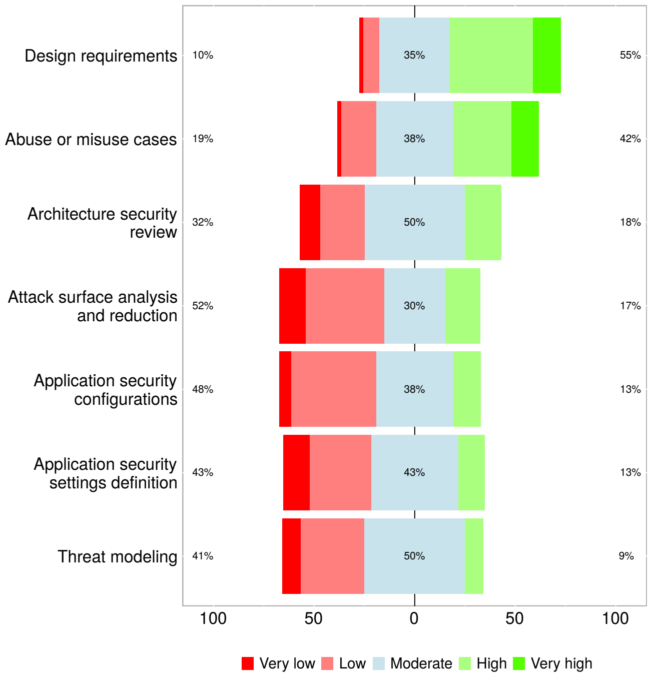

Figure 5: Perceived impact at design phase

The results regarding the eight surveyed implementation phase activities are illustrated in Fig. 6 and Fig. 7. Coding standards are the most frequently used activity during the implementation phase. These are also perceived to have a high impact upon security. The observations are not surprising because coding standards are also an integral

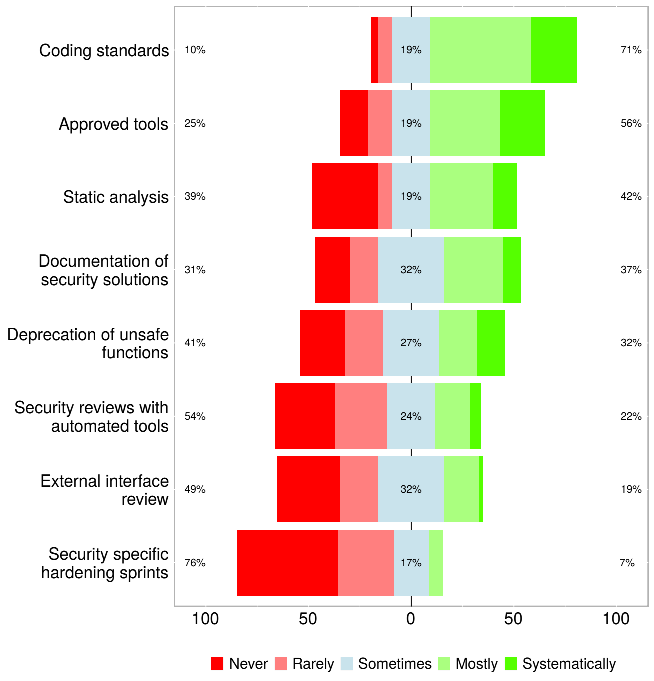

Figure 6: Use at implementation phase

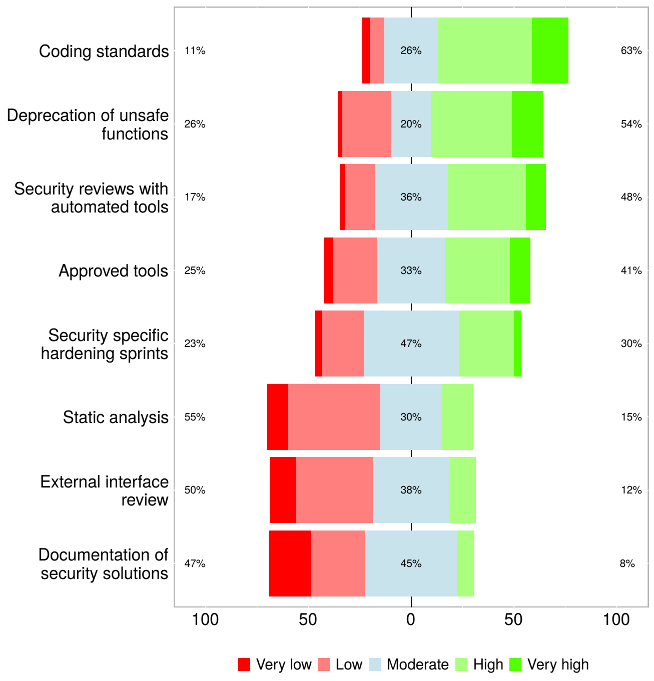

Figure 7: Perceived impact at implementation phase

part of common agile software development principles.

The tools used for development are frequently agreed upon and specified security documentation, as are the types of performed static analyses. These observations seem logical in the sense that coding standards, use of common tools, and static analysis are frequently used irrespective of whether the context is explicitly related to security. The same explanation may apply to the low perceived security impact of static analysis and documentation. In terms of the former, the result supports other recent surveys [47], possibly also reflecting the limited capability of many static analysis tools for detecting security weaknesses [37, 54]. In terms of the latter, the low perceived impact is expected due to the sample's inherent bias toward software developers.

There is a substantial mismatch between the use and perceived impact of a few particular implementation-time activities. In particular, security reviews with automated tools and security-specific hardening sprints receive only infrequent use, despite both of these activities are perceived to have a relatively high impact upon security. Regulation aspects may again partially explain these results. For instance, security hardening sprints have been promoted in the safety context as a way to ensure regulatory compliance [17]. For agile projects dealing with less strict compliance requirements, however, the use of hardening sprints may constrain the development akin to quality gates, especially in case these do not relate to auditing or one-shot verification. That is, these hardening sprints may introduce "security gate", contradictory to the agile work. Dedicated security sprints consume valuable development time, which needs to pay off for example in the form of passing a security audit. The answer to this question is not straightforward: it may be that security-specific hardening sprints are either unnecessary, or they are considered difficult to apply to agile work without breaking the flexible practices.

4.2.4. Verification phase

At the security testing phase, various security validation and verification activities are performed to locate any weaknesses in the implementation, documentation or configurations that could introduce exploitable security vulnerabilities. The activities performed in this phase also aim to validate and produce proof that the implementation complies with the security requirements.

In iterative software development, the potentially production-ready release candidates are selected as a result of the verification and validation processes. The surveyed activities behind these processes are shown in Fig. 8 and Fig. 9.

In general, the use is heavily balanced toward testing. With code reviews thrown in, the most frequently used verification activities include the use of automated testing tools, security specific test cases, and, to a lesser extent, penetration testing. These results underline the test-driven ethos of agile software development as well as the contemporary trend toward automation.

While testing is an extensively used software and security development practice, low use of systematic test plans is a trend noted in related software industry research [57]. Although reliance upon automation and extended use of automated security testing tools is a positive

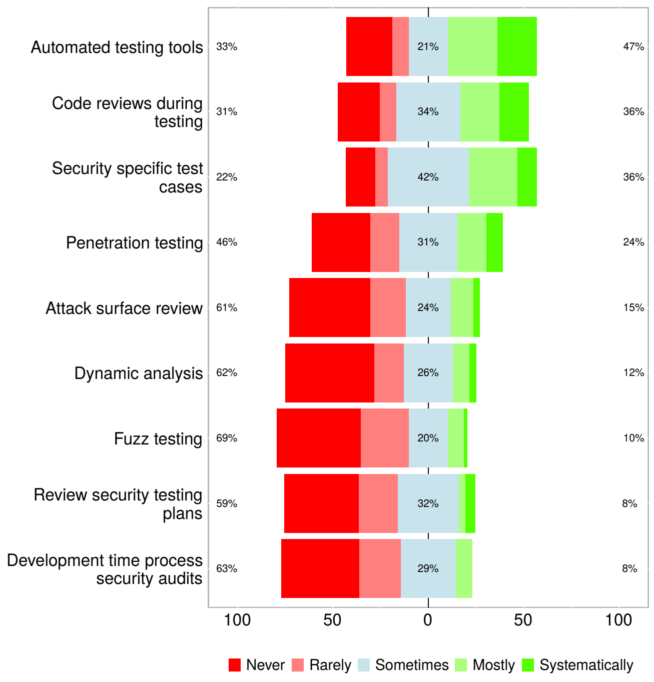

Figure 8: Use at verification phase

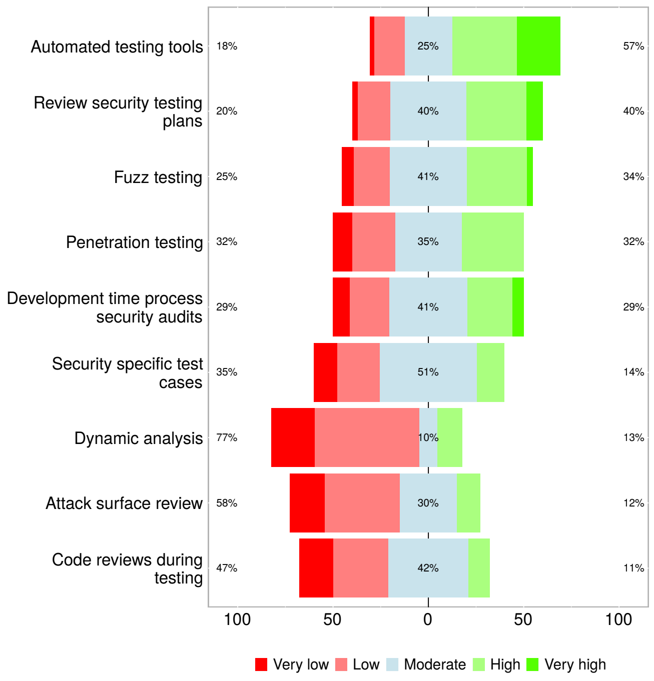

Figure 9: Perceived impact at verification phase

security trend, the significantly low rate of reviewing security testing plans possibly indicates a straightforward absence of security test plans, supported by these general findings.

Although the survey design does not allow to draw definite conclusions, it seems that many of the commonly used activities during the verification phase align with the implementation-time activities such as static analysis. Such alignment is also a typical problem because traditional security (safety) engineering emphasizes post-development testing, whereas agile practices concentrate on development-time testing [23]. To some extent, this potential problem is also visible in the results shown: the auditing of the development-time testing practices is the least used activity. That said, it should be recalled that formal audits have been a rare requirement in the projects the respondents have worked with.

It is also worth to emphasize that not all testing techniques are equally used: dynamic (run-time) analysis and fuzzing are only rarely used by the respondents and their projects. Limited use of dynamic analysis has been reported also in other surveys [47]. Furthermore, dynamic analysis is perceived to have only a low impact upon security, whereas fuzz testing is seen to have a high impact. Development efficiency offers a plausible explanation for these results. For instance, dynamic analysis requires heavy human involvement and strong technical skills [37]. Likewise, building and configuring a fuzzing environment requires a substantial amount of time.

A typical fuzz testing period is 24 hours for each tested software instance [33]. As fuzz testing tools keep finding crashes also after that, fuzzing should be applied for long periods of time or even continuously. Thus, the required effort and the time restraints may again conflict with typical agile work flow. Such a conflict may also explain the misalignment observed: even though the respondents perceive fuzzing as an efficient way to improve software security, they also acknowledge the limited use of this testing technique.

4.2.5. Release phase

In security development life cycle, the release phase activities prepare the software for release and ensure its secure maintenance. Although the increasing use of continuous integration and continuous delivery practises have somewhat blurred the boundaries of the life cycle models, many issues related to maintenance must be still specifically addressed. Regulatory requirements also constrain the trend toward full automation of release engineering [36]. Thus, the security activities performed at this phase include auditing the product to be released, producing security assurance and documentation for maintenance and operations, and ensuring that different operational security mechanisms are in place. The results on the surveyed release-time activities are shown in Figs. 10 and 11.

The most used activity at the release phase relates to work required to ensure that host and network security are in place. Given the contemporary deployment practices, this work presumably involves also addressing the security questions related to cloud computing platforms [82]. Given that most of the respondents are dealing with web applications and cloud platforms (see Subsection 4.1.1), it

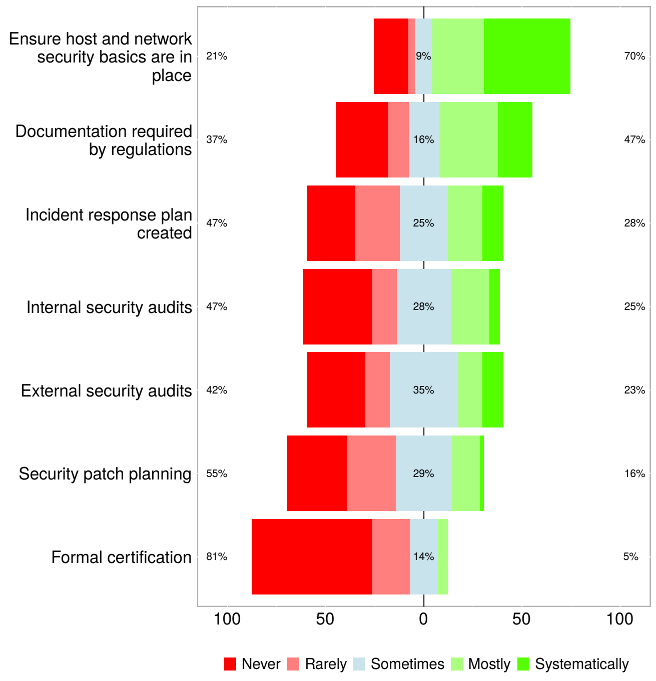

Figure 10: Use at release phase

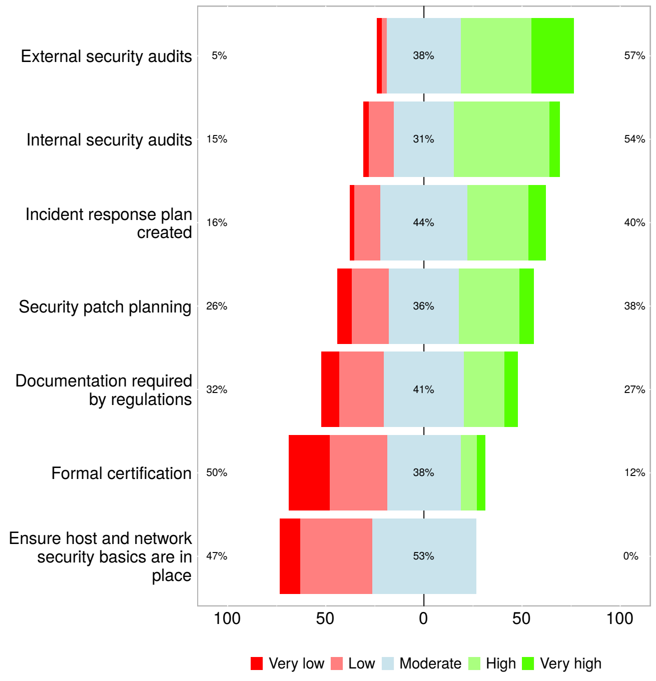

Figure 11: Perceived impact at release phase

may also be that the requirements for host and network security can be replicated across multiple projects. These can be further seen as prerequisites for other security requirements [61]. This fundamental nature may also explain the apparent misalignment: none of the respondents perceive host and network security basics as having a high or a very high security impact. A competing explanation for the observation would be that general network security aspects are over-emphasized at the expense of security of the software itself.

The survey questionnaire contained also a specific question about whether the respondents' organizations had at some point obtained at least one security certificate through a formal auditing process. To this question, 18% of the respondents answered positively. The Finnish national standards VAHTI and KATAKRI were both among the most frequently audited ones. As seen in Figure 11, the respondents found the audits to have a high positive impact on security. Furthermore, activities inherent to security audits, such as various reviews and security tests, were considered among the most impactful throughout the surveyed phases.

The respondents also commonly produce documents required by regulations. This observation indicates that security engineering practices can be useful in augmenting the typically minimal documentation produced in agile development. This result can be combined with the relatively frequent documentation activities during the earlier development phases. Agile software development has been criticized for "ad hoc, inaccurate, incomplete, or non-existing documentation" [16]. As such, the results presented do not seem to support this argument, at least when security matters are concerned.

On the side of rarely used release-time activities, various audits and certifications were only seldom used in the projects the respondents have worked in. As these are typical activities imposed by regulations and other external constraints, the explanation is again partially related to the sample characteristics. On the other hand, these observations can be also interpreted to reflect a generally low demand for audit-related security assurance. However, when used, both internal and external security audits are perceived to have a substantial impact upon security. The release-time activities further exhibit the general mismatch between the frequency of use and the perceived security impact of software security activities.

5. Discussion and analysis

Filling a novel gap in software engineering research, the survey produces an interesting insight into the way software practitioners organize their security-related work in an agile software development setting. This chapter discusses these findings and their implications to both industry and the research community.

5.1. Key results

The iterative and incremental agile software development is often seen as an ill-match to many security engineering models that require up-front design that may diminish agility. Given this motivation, the paper presented an industry survey on the use and perceived impact of various security engineering activities in the context of agile software developments.

Two research questions were asked. The answers (A) to the RQs can be summarized as follows:

RQ1: To what extent are security engineering activities utilized in the context of agile software development?

A1: The two activity types are used in conjunction with each other. Agile practices concentrate on requirements engineering, implementation, and extensive testing. In contrast, much of security engineering work is done in the verification phase.

RQ2: What is the perceived security impact of security engineering activities in agile software development?

A2: The results prompt two main observations: (1) The earlier the security activity is performed, the more effective it is perceived to be. These include requirement, design and implementation phase activities; (2) the use of automated testing tools and specific security testing methods are deemed most effective security practices together with release-time security audits. This is a clear indication of a strong preference to ensure the security and secure implementation before deployment, enabling cost-efficient modifications to the software and its security configurations.

The selected research questions describe the targeted population of software security professionals. The measured activities were gathered from security frameworks, standards and life cycle models. This also removed much of the relevance of analyzing e.g. the use of individual activities within the life cycle phases: the results describe the industry best practices in a focused target group, and thus provide a benchmark for general population. Inferring patterns for general population of software developers from these results would provide very little benefit.

5.2. Implications for industry

The survey and the research questions draw an interesting the field of software security practitioners: the selection of practices, and to a degree, the factors guiding the security work. First of all, the respondents have thoroughly embraced the agile practices while combining them to a variety of security engineering practices. Agile practitioners tend to use simpler security techniques, which are concentrated on the early phases of the SSDLC.

The demand for software security and assurance of security's existence is increasing from the perspectives of the regulators and the software users. This is best achieved by introducing security elements into the software from the very beginning of its life cycle [55]. This calls for rigorous management practices and unified security metrics usable for several types of software products and projects. On tool support for effective ways to bind together the security requirements, design, implementation and verification activities would provide means for e.g. dependency management and vulnerability tracking in software modules and connected components.

Regulation is a main driving force in the security work, although not the only one. The results imply a need for improvement in the mechanisms how security regulation imposes requirements to software development. Especially the security assurance items regarding security policies and security enforcement mechanisms have a decreased level of usage despite their perceived effectiveness, implying bad conformance with software development processes. Formal compliance requirements have strayed quite far from the practical work done in day-to-day software development. This has an implication that certain process-related security improvement practices, although effective, remain mostly unused in mainstream software development projects. It is understandable, however, that formal or semi-formal process and documentation reviews are performed only under direct regulatory requirements.

5.3. Implications for research

Many of the security engineering activities in the Software Security Life Cycle model are used also in agile software development. However, neither the SSDLC nor other models are usually adopted wholesale. These typically act as security engineering frameworks from which practitioners pick and choose the practices deemed most suitable for their work. Another important point is the poor alignment between the use of the activities and their perceived impact upon software security. While this misalignment can be attributed to several possible factors, there is a dual implication from the mismatch. On one hand, software development organizations should be more proactive in updating their processes; on the other, the research community working on software security development models should pay more attention to their adaptability.

The adaptability question can be considered also at the level of individual software security engineering activities. To this end, a good problem for further research would be the identification of activities that are synergic. In other words, some particular security engineering activity might be used more efficiently as a part of a general software development activity. For instance, elicitation of security requirements might be reasonably attainable as a part of Scrum's backlog formation. Likewise, coding standards and code reviews are in line with core agile principles [39]. Furthermore, static and dynamic analysis should be integrated into code review processes instead of being treated as isolated activities [45]. The reverse direction should be also considered; there are also certain security engineering activities that have a poor synergy with traditional agile software development activities.

By comparing synergic and non-synergic security engineering activities, it might be also possible to better understand the impact of security engineering on software development efficiency. The relationship between security and efficiency is a complex one, however. Practical evaluation requires that a software development organization has knowledge about the level of security attained and the development efficiency attainable via the use of a particular set of security engineering activities. A good question for further research would be the means by which the evaluation on the security-efficiency nexus could be carried out.

Finally, it is worth remarking a positive correlation between all of the security engineering activities and a sum variable formed based on the arithmetic mean of the sixteen agile work practices in Fig. 1. This is a tentative but important observation for further research. In other words, increasing agility is associated with increasing use of software security engineering activities, and the other way around. These are generally interesting preliminary findings because a common folk wisdom would entail negative correlations; that less agile or more rigid organizations would be more likely to use security engineering activities outside of the standard agile toolbox, or even as parallel processes [60]. In contrast, the results indicate that organizational rigidity is associated with decreasing use of the activities surveyed.

6. Threats to validity

Several measures were taken in the preparation, design, and execution of the survey, and the analysis of the data, to mitigate or to avoid the threats to validity of the research. The security activities were collected from several software security frameworks and standards, and the software activities and artifacts from an earlier study performed among software developers in Finland.

The research instrument, an open and anonymous questionnaire, was assessed to pose certain challenges to the independence of the data collected. The challenges were identified as follows: (1) inability to identify the respondents' organization, project, or customer they are working on; (2) acknowledging that the respondents share the same background, have same or very similar education, and are very likely to share the same security awareness training; (3) acknowledging that the respondents form a group of software security specialists, that work in same homogeneous regulatory environment, share their experiences, and draw from the same security influences [cf. 21].

To further mitigate threats to reliability, a portion of the respondents were individually picked from software and security companies, and the questionnaire itself piloted with selected subject matter experts (N=20). The questionnaire was designed to mitigate the misunderstandings the respondents may have about the security engineering activities via clarifying clauses and logical question grouping. To the degree the usability of the survey design allowed, control questions were used to verify the answers. However, the substantial amount of security activities (40) may have caused some misinterpretations, posing a possible threat to reliability. The data was also purged of any outliers and checked for possible vandalism, which led to the removal of one response.

The focused sample, the strict scoping of the survey questions, and further limiting of the surveyed projects to the agile software development projects with specific security activities, the randomness of the sample was likely to be reduced. The respondents were selected from an extensive precompiled list of Finnish software companies, with the respondents representing a subset of those organizations. As the respondents reported a substantial level of use of organizations' self-designed security constraints, a possible threat to internal validity was recognized in the form of organizational clustering.

The respondents were experts in software security. Thus, generalizing the results from the observed sample to all software engineers was deemed to produce potentially misleading results. To avoid this threat to external validity, a decision was made to present the results using descriptive statistics. The results provide the general software developer population a clear and concise picture about how the experts conduct software security engineering, and which activities they find to be most impactful.

7. Concluding remarks

The survey, performed among software practitioners working with Finnish software development companies, gives a positive view of the level of professionalism and security capabilities of the respondents. The respondents give the appearance of taking security work seriously, and they genuinely worry about the security of the software products they develop.

This survey opens some quite interesting research avenues. It was also designed to give a clear baseline for comparison to coming surveys. The effect of various regulative security practices should be thoroughly examined, as the body of national and international security regulation is constantly growing in response to increasing security concerns. Effectiveness of security work in software development gives a competitive edge to the early adapters, and to those who best manage the integration of the processes. Qualitative approaches drilling into these crucial topics should provide immediately useful and interesting results for both security researchers and practitioners.

References

[1] Anderson, R., 2008. Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd Edition. Wiley, Indianapolis.

[2] Avizienis, A., Laprie, J. ., Randell, B., Landwehr, C., Jan 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1 (1), 11–33.

[3] Ayalew, T., Kidane, T., Carlsson, B., 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. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 139–153.
URL `http://dx.doi.org/10.1007/978-3-642-41488-6_10`

[4] Baca, D., Carlsson, B., 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, pp. 149–158.
URL `http://doi.acm.org/10.1145/1987875.1987900`

[5] Bartsch, S., Aug 2011. Practitioners' perspectives on security in agile development. In: 2011 Sixth International Conference on Availability, Reliability and Security. pp. 479–484.

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

[7] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al., 2001. Manifesto for agile software development. Online at http://www.agilemanifesto.org.

[8] Bell, L., Brunton-Spall, M., Smith, R., Bird, J., 2017. Agile Application Security: Enabling Security in a Continuous Delivery Pipeline. O'Reilly Media, Inc.

[9] Bellomo, S., Kruchten, P., Nord, R. L., Ozkaya, I., 2014. How to agilely architect an agile architecture. Cutter IT Journal 27 (2), 12–17.

[10] Ben-Othmane, L., Angin, P., Weffers, H., Bhargava, B., Nov 2014. Extending the agile development process to develop acceptably secure software. IEEE Transactions on Dependable and Secure Computing 11 (6), 497–509.

[11] Beznosov, K., Kruchten, P., 2004. Towards agile security assurance. In: NSPW '04 Proceedings of the 2004 workshop on New security paradigms. pp. 47–54.

[12] Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K., Kruchten, P., 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, pp. 11–18.
URL `http://doi.acm.org/10.1145/1137627.1137631`

[13] Chivers, H., Paige, R. F., Ge, X., 2005. Agile security using an incremental security architecture. In: Baumeister, H., Marchesi, M., Holcombe, M. (Eds.), Extreme Programming and Agile Processes in Software Engineering: 6th International Conference, XP 2005, Sheffield, UK, June 18-23, 2005. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 57–65.
URL `http://dx.doi.org/10.1007/11499053_7`

[14] Cockburn, A., Williams, L., 2000. The costs and benefits of pair programming. Extreme programming examined 8, 223–247.

[15] Conboy, K., Fitzgerald, B., Golden, W., 2005. Agility in information systems development: A three-tiered framework. In: Baskerville, R. L., Mathiassen, L., Pries-Heje, J., DeGross, J. I. (Eds.), Business Agility and Information Technology Diffusion: IFIP TC8 WG 8.6 International Working Conference May 8–11, 2005, Atlanta, Georgia, U.S.A. Springer US, Boston, MA, pp. 35–49.
URL `https://doi.org/10.1007/0-387-25590-7_3`

[16] Drury-Grogan, M. L., Conboy, K., Acton, T., 2017. Examining decision characteristics & challenges for agile software development. Journal of Systems and Software 131, 248–265.

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

[18] Ge, X., Paige, R., Polack, F., Brooke, P., 2007. Extreme programming security practices. In: Concas, G., Damiani, E., Scotto, M., Succi, G. (Eds.), Agile Processes in Software Engineering and Extreme Programming. Vol. 4536 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 226–230.
URL `http://dx.doi.org/10.1007/978-3-540-73101-6_42`

[19] Geer, D., June 2010. Are companies actually using secure development life cycles? Computer 43 (6), 12–16.

[20] Ghani, I., Arbain, A. F. B., Oueslati, H., Rahman, M. M., Ben-Othmane, L., Jan. 2016. Evaluation of the challenges of developing secure software using the agile approach. Int. J. Secur. Softw. Eng. 7 (1), 17–37.
URL `http://dx.doi.org/10.4018/IJSSE.2016010102`

[21] Grawitch, M. J., Munz, D. C., 2004. Are your data nonindependent? a practical guide to evaluating nonindependence and within-group agreement. Understanding Statistics 3 (4), 231–257.

[22] Hamid, B., Weber, D., 2018. Engineering secure systems: Models, patterns and empirical validation. Computers & Security 77, 315–348.

[23] Heeager, L. T., Nielsen, P. A., 2018. A conceptual model of agile software development in a safety-critical context: A systematic literature review. Information and Software Technology 103, 22–39.

[24] Holvitie, J., Licorish, S. A., Spínola, R. O., Hyrynsalmi, S., MacDonell, S. G., Mendes, T. S., Buchan, J., Leppänen, V., 2018. Technical debt and agile software development practices and processes: An industry practitioner survey. Information and Software Technology 96, 141–160.

[25] Howard, M., Lipner, S., 2006. The security development lifecycle. Vol. 8. Microsoft Press Redmond.

[26] ICS-CERT, 2016. Recommended Practice: Improving Industrial Control System Cybersecurity with Defense-in-Depth Strategies. U.S. Homeland Security.
URL `https://ics-cert.us-cert.gov/sites/default/files/recommended_practices/NCCIC_ICS-CERT_Defense_in_Depth_2016_S508C.pdf`

[27] IEEE, 2018. Avoiding the top 10 software security design flaws.
URL `https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf`

[28] ISO/IEC Sstandard 27034-1:2011, 2011. Information technology — Security techniques — Application security — Part 1: Overview and concepts. ISO/IEC.

[29] ISO/IEC standard 15026-4:2012, 2012. Systems and software engineering – Systems and software assurance – Part 4: Assurance in the life cycle. ISO/IEC.

[30] ISO/IEC standard 15408-1:2009, 2014. Information technology - Security techniques - Evaluation criteria for IT security, 3rd Edition. ISO/IEC.

[31] ISO/IEC standard 21827:2008, 2008. Information Technology – Security Techniques – Systems Security Engineering – Capability Maturity Model (SSE-CMM), 2nd Edition. ISO/IEC.

[32] Kasauli, R., Knauss, E., Kanagwa, B., Nilsson, A., Calikli, G., Aug 2018. Safety-critical systems and agile development: A mapping study. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 470–477.

[33] Klees, G., Ruef, A., Cooper, B., Wei, S., Hicks, M., Aug. 2018. Evaluating Fuzz Testing. ArXiv e-prints.

[34] Kongsli, V., 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, pp. 805–808.
URL `http://doi.acm.org/10.1145/1176617.1176727`

[35] Kuhn, R., Raunak, M., Kacker, R., 2018. Can reducing faults prevent vulnerabilities? Computer 51 (7), 82–85.

[36] Laukkarinen, T., Kuusinen, K., Mikkonen, T., 2018. Regulated software meets devops. Information and Software Technology 97, 176 – 178.
URL `http://www.sciencedirect.com/science/article/pii/S0950584918300144`

[37] Li, J., Zhao, B., Zhang, C., Jun 2018. Fuzzing: a survey. Cybersecurity 1 (1), 6.
URL `https://doi.org/10.1186/s42400-018-0002-y`

[38] Licorish, S., Holvitie, J., Spínola, R., Hyrynsalmi, S., Buchan, J., Mendes, T., MacDonnell, S., Leppänen, V., 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, pp. 369–372.

[39] Martin, R. C., 2009. Clean code: a handbook of agile software craftsmanship. Pearson Education.

[40] McDermott, J., Dec 2001. Abuse-case-based assurance arguments. In: Seventeenth Annual Computer Security Applications

Conference. pp. 366–374.
[41] McDermott, J., Fox, C., Dec 1999. Using abuse case models for security requirements analysis. In: Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99). pp. 55–64.
[42] McGraw, G., 2006. Software Security: Building Security In. Addison-Wesley Professional.
[43] Microsoft, 2019. Agile development using microsoft security development lifecycle.
[44] Middleton, A., B. E. P. M. o. b. o. t. D. S., 2014. Finding people who will tell you their thoughts on genomics—recruitment strategies for social sciences research. Journal of Community Genetics 5, 291–302.
[45] Mitropoulos, D., Spinellis, D., 2017. Fatal Injection: A Survey of Modern Code Injection Attack Countermeasures. PeerJ Computer Science 3 (e136), 1–40.
[46] Morrison, P., Moye, D., Pandita, R., Williams, L., 2018. Mapping the field of software life cycle security metrics. Information and Software Technology 102, 146 – 159.
URL `http://www.sciencedirect.com/science/article/pii/S095058491830096X`
[47] Nembhard, F. D., Carvalho, M. M., Eskridge, T. C., 2019. Towards the application of recommender systems to secure coding. EURASIP Journal on Information Security 9, 1–24.
[48] Nerur, S., Mahapatra, R., Mangalaraj, G., May 2005. Challenges of migrating to agile methodologies. Commun. ACM 48 (5), 72–78.
URL `http://doi.acm.org/10.1145/1060710.1060712`
[49] NIST, 2018. Source code security analyzers.
[50] Nurdiani, I., Börstler, J., Fricker, S., Petersen, K., Chatzipetrou, P., 2019. Understanding the order of agile practice introduction: Comparing agile maturity models and practitioners' experience. Journal of Systems and Software 156, 1–20.
[51] OWASP, 2018. Owasp top 10 application security risks.
URL `https://www.owasp.org/index.php/Top_10-2017_Top_10`
[52] OWASP SAMM, 2019. Software assurance maturity model.
[53] Oyetoyan, T. D., Cruzes, D. S., Jaatun, M. G., Aug 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). pp. 548–555.
[54] Oyetoyan, T. D., Milosheska, B., Grini, M., Cruzes, D. S., 2018. Myths and facts about static application security testing tools: An action research at telenor digital. In: Proceedings of the 19th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP 2018). Springer, pp. 86–103.
[55] Phillips, D. M., Mazzuchi, T. A., Sarkani, S., 2018. An architecture, system engineering, and acquisition approach for space system software resiliency. Information and Software Technology 94, 150–164.
[56] Poth, A., Sasabe, S., Mas, A., Mesquida, A., 2018. Lean and agile software process improvement in traditional and agile environments. Journal of Software: Evolution and Process 0 (0).
[57] Rahikkala, J., Hyrynsalmi, S., Leppänen, V., 2015. Accounting testing in software cost estimation: A case study of the current practice and impacts. In: SPLST. pp. 61–75.
[58] Rice, T., Brown-White, J., Skinner, T., Ozmore, N., Carlage, N., Poland, W., Heitzman, E., Dhillon, D., 2018. Fundamental practices for secure software development 3rd edition. In: undamental practices for secure software development. SAFECode, p. 38.
[59] Rindell, K., Hyrynsalmi, S., Leppänen, V., 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, pp. 74:1–74:10.
URL `http://doi.acm.org/10.1145/3098954.3103170`
[60] Rindell, K., Hyrynsalmi, S., Leppänen, V., 2017. Case study of agile security engineering: Building identity management for a government agency. International Journal of Secure Software Engineering 8, 43–57.
[61] Rindell, K., Ruohonen, J., Hyrynsalmi, S., 2018. Surveying secure software development practices in finland. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. ARES 2018. ACM, New York, NY, USA, pp. 6:1–6:7.
URL `http://doi.acm.org/10.1145/3230833.3233274`
[62] SANS, 2011. CWE/SANS top 25 most dangerous software errors.
URL `https://www.sans.org/top25-software-errors`
[63] Scandariato, R., Wuyts, K., Joosen, W., 2015. A descriptive study of microsoft's threat modeling technique. Requirements Engineering 20, 163–180.
[64] Schwaber, K., Beedle, M., 2002. Agile Software Development with Scrum, 1st Edition. Prentice Hall PTR, Upper Saddle River, NJ, USA.
[65] Schweigert, T., Vohwinkel, D., Korsaa, M., Nevalainen, R., Biro, M., 2014. Agile maturity model: Analysing agile maturity characteristics from the SPICE perspective. Journal of Software: Evolution and Process 26 (5), 513–520.
[66] Séguin, N., Tremblay, G., Bagane, H., 2012. Agile principles as software engineering principles: An analysis. In: Wohlin, C. (Ed.), Agile Processes in Software Engineering and Extreme Programming. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–15.
[67] Silva, F. S., Soares, F. S. F., Peres, A. L., de Azevedo, I. M., Vasconcelos, A. P. L., Kamei, F. K., de Lemos Meira, S. R., 2015. Using cmmi together with agile software development: A systematic review. Information and Software Technology 58, 20 – 43.
URL `http://www.sciencedirect.com/science/article/pii/S0950584914002110`
[68] Stavru, S., 2014. A critical examination of recent industrial surveys on agile method usage. Journal of Systems and Software 94, 87 – 97.
[69] Such, J. M., Gouglidis, A., Knowles, W., Misra, G., Rashid, A., 2016. Information assurance techniques: Perceived cost effectiveness. Computers & Security 60, 117 – 133.
URL `http://www.sciencedirect.com/science/article/pii/S0167404816300311`
[70] Synopsys Software Integrity Group, 2017. The building security in maturity model.
URL `https://www.bsimm.com/`
[71] Theisen, C., Munaiah, N., Al-Zyoud, M., Carver, J. C., Andrew Meneelyb, L. W., 2018. Attack surface definitions: A systematic literature review. Information and Software Technology 104, 94–103.
[72] Tøndel, I. A., Jaatun, M. G., Meland, P. H., Jan 2008. Security requirements for the rest of us: A survey. IEEE Software 25 (1), 20–27.
[73] Tsipenyuk, K., Chess, B., McGraw, G., 2005. Seven pernicious kingdoms: A taxonomy of software security errors. IEEE Security & Privacy 3 (6), 81–84.
[74] Turner, R., Jain, A., 2002. Agile meets cmmi: Culture clash or common cause? In: Wells, D., Williams, L. (Eds.), Extreme Programming and Agile Methods — XP/Agile Universe 2002. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 153–165.
[75] Türpe, S., Poller, A., 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, pp. 34–49.
[76] VAHTI, 2015. VAHTI-ohje (trans. VAHTI instruction).
URL `http://www.vahtiohje.fi/web/guest`
[77] VersionOne, 2018. 12th annual state of agile survey.
[78] Viega, J., McGraw, G., 2002. Building Secure Software: How to Avoid Security Problems the Right Way, 1st Edition. Addison-Wesley.
[79] Wäyrynen, J., Bodén, M., Boström, G., 2004. Security engineering and extreme programming: An impossible marriage? In: Zannier, C., Erdogmus, H., Lindstrom, L. (Eds.), Extreme

Programming and Agile Methods - XP/Agile Universe 2004: 4th Conference on Extreme Programming and Agile Methods, Calgary, Canada, August 15-18, 2004. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 117–128.
URL `http://dx.doi.org/10.1007/978-3-540-27777-4_12`

[80] Williams, L., Kessler, R. R., Cunningham, W., Jeffries, R., Jul 2000. Strengthening the case for pair programming. IEEE Software 17 (4), 19–25.

[81] Williams, L., McGraw, G., Migues, S., Sep. 2018. Engineering security vulnerability prevention, detection, and response. IEEE Software 35 (5), 76–80.

[82] Younas, M., Jawawi, D. N. A., Ghani, I., Fries, T., Kazmi, R., 2018. Agile development in the cloud computing environment: A systematic review. Information and Software Technology 103, 142–158.