Kalle Rindell

Development of Secure Software

Rationale, Standards and Practices

Turku Centre *for* Computer Science

TUCS Dissertations
No 239, June 2019

Development of Secure Software

Rationale, Standards and Practices

Kalle Rindell

To be presented, with the permission of the Faculty of Science and Engineering of the University of Turku, for public criticism in Auditorium XXI on June 17, 2019, at 12 noon.

University of Turku
Department of Future Technologies
Vesilinnantie 5, 20500 Turku, Finland

2019

Supervisors

Prof. Ville Leppänen, Ph.D.
Department of Future Technologies
University of Turku
Finland

Asst. Prof. Sami Hyrynsalmi, D.Sc. (Tech.)
Computing Sciences
Tampere University
Finland

Reviewers

Prof. Martin Gilje Jaatun, dr.philos.
Software Engineering, Safety and Security
SINTEF Digital
Norway

Prof. Alexander Horst Norta, Ph.D.
Department of Software Science
Tallinn University of Technology
Estonia

Opponent

Prof. Pekka Abrahamsson, Ph.D.
Faculty of Information Technology
University of Jyväskylä
Finland

ISBN 978-952-12-3833-8
ISSN 1239-1883

Ystäville ja sukulaisille.

—

For friends and relatives.

Abstract

The society is run by software. Electronic processing of personal and financial data forms the core of nearly all societal and economic activities, and concerns every aspect of life. Software systems are used to store, transfer and process this vital data. The systems are further interfaced by other systems, forming complex networks of data stores and processing entities. This data requires protection from misuse, whether accidental or intentional. Elaborate and extensive security mechanisms are built around the protected information assets. These mechanisms cover every aspect of security, from physical surroundings and people to data classification schemes, access control, identity management, and various forms of encryption.

Despite the extensive information security effort, repeated security incidents keep compromising our financial assets, intellectual property, and privacy. In addition to the direct and indirect cost, they erode the trust in the very foundation of information security: availability, integrity, and confidentiality of our data. Lawmakers at various national and international levels have reacted by creating a growing body of regulation to establish a baseline for information security. Increased awareness of information security issues has led to extend this regulation to one of the core issues in secure data processing: security of the software itself.

Information security contains many aspects. It is generally classified into organizational security, infrastructure security, and application security. Within application security, the various security engineering processes and techniques utilized at development time form the discipline of *software security engineering*. The aim of these security activities is to address the software-induced risk toward the organization, reduce the security incidents and thereby lower the lifetime cost of the software. Software security engineering manages the software risk by implementing various security controls right into the software, and by providing security assurance for the existence of these controls by verification and validation.

A software development process has typically several objectives, of which security may form only a part. When security is not expressly prioritized, the development organizations have a tendency to direct their resources to

the primary requirements. While producing short-term cost and time savings, the increased software risk, induced by a lack of security and assurance engineering, will have to be mitigated by other means. In addition to increasing the lifetime cost of software, unmitigated or even unidentified risk has an increased chance of being exploited and cause other software issues.

This dissertation concerns security engineering in agile software development. The aim of the research is to find ways to produce secure software through the introduction of security engineering into the agile software development processes. Security engineering processes are derived from extant literature, industry practices, and several national and international standards. The standardized requirements for software security are traced to their origins in the late 1960s, and the alignment of the software engineering and security engineering objectives followed from their original challenges to the current agile software development methods. The research provides direct solutions to the formation of security objectives in software development, and to the methods used to achieve them. It also identifies and addresses several issues and challenges found in the integration of these activities into the development processes, providing directly applicable and clearly stated solutions for practical security engineering problems.

The research found the practices and principles promoted by agile and lean software development methods to be compatible with many security engineering activities. Automated, tool-based processes and the drive for efficiency and improved software quality were found to directly support the security engineering techniques and objectives. Several new ways to integrate software engineering into agile software development processes were identified. Ways to integrate security assurance into the development process were also found, in the form of security documentation, analyses, and reviews. Assurance artifacts can be used to improve software design and enhance quality assurance. In contrast, detached security engineering processes may create security assurance that serves only purposes external to the software processes. The results provide direct benefits to all software stakeholders, from the developers and customers to the end users.

Security awareness is the key to more secure software. Awareness creates a demand for security, and the demand gives software developers the concrete objectives and the rationale for the security work. This also creates a demand for new security tools, processes and controls to improve the efficiency and effectiveness of software security engineering. At first, this demand is created by increased security regulation. The main pressure for change will emanate from the people and organizations utilizing the software: security is a mandatory requirement, and software must provide it. This dissertation addresses these new challenges. Software security continues to gain importance, prompting for new solutions and research.

Tiivistelmä

Ohjelmistot ovat keskeinen osa yhteiskuntamme perusinfrastruktuuria. Merkittävä osa sosiaalisesta ja taloudellisesta toiminnastamme perustuu tiedon sähköiseen käsittelyyn, varastointiin ja siirtoon. Näitä tehtäviä suorittamaan on kehitetty merkittävä joukko ohjelmistoja, jotka muodostavat mutkikkaita tiedon yhteiskäytön mahdollistavia verkostoja. Tiedon suojaamiseksi sen ympärille on kehitetty lukuisia suojamekanismeja, joiden tarkoituksena on estää tiedon väärinkäyttö, oli se sitten tahatonta tai tahallista. Suojausmekanismit koskevat paitsi ohjelmistoja, myös niiden käyttöympäristöjä ja käyttäjiä sekä itse käsiteltävää tietoa: näitä mekanismeja ovat esimerkiksi tietoluokittelut, tietoon pääsyn rajaaminen, käyttäjäidentiteettien hallinta sekä salaustekniikat.

Suojaustoimista huolimatta tietoturvaloukkaukset vaarantavat sekä liiketoiminnan ja yhteiskunnan strategisia tietovarantoj että henkilökohtaisia tietojamme. Taloudellisten menetysten lisäksi hyökkäykset murentavat luottamusta tietoturvan kulmakiviin: tiedon luottamuksellisuuteen, luotettavuuteen ja sen saatavuuteen. Näiden tietoturvan perustusten suojaamiseksi on laadittu kasvava määrä tietoturvaa koskevia säädöksiä, jotka määrittävät tietoturvan perustason. Lisääntyneen tietoturvatietoisuuden ansiosta uusi säännöstö on ulotettu koskemaan myös turvatun tietojenkäsittelyn ydintä, ohjelmistokehitystä.

Tietoturva koostuu useista osa-alueista. Näitä ovat organisaatiotason tietoturvakäytännöt, tietojenkäsittelyinfrastruktuurin tietoturva, sekä tämän tutkimuksen kannalta keskeisenä osana ohjelmistojen tietoturva. Tähän osa-alueeseen sisältyvät ohjelmistojen kehittämisen aikana käytettävät tietoturvatekniikat ja -prosessit. Tarkoituksena on vähentää ohjelmistojen organisaatioille aiheuttamia riskejä, tai poistaa ne kokonaan. Ohjelmistokehityksen tietoturva pyrkii pienentämään ohjelmistojen elinkaarikustannuksia määrittämällä ja toteuttamalla tietoturvakontrolleja suoraan ohjelmistoon itseensä. Lisäksi kontrollien toimivuus ja tehokkuus osoitetaan erillisten verifiointi- ja validointimenetelmien avulla.

Tämä väitöskirjatutkimus keskittyy tietoturvatyöhön osana iteratiivista ja inkrementaalista ns. ketterää (agile) ohjelmistokehitystä. Tutkimuksen

tavoitteena on löytää uusia tapoja tuottaa tietoturvallisia ohjelmistoja liittämällä tietoturvatyö kiinteäksi osaksi ohjelmistokehityksen prosesseja. Tietoturvatyön prosessit on johdettu alan tieteellisestä ja teknillisestä kirjallisuudesta, ohjelmistokehitystyön vallitsevista käytännöistä sekä kansallisista ja kansainvälisistä tietoturvastandardeista. Standardoitujen tietoturvavaatimusten kehitystä on seurattu aina niiden alkuajoilta 1960-luvulta lähtien, liittäen ne ohjelmistokehityksen tavoitteiden ja haasteiden kehitykseen: nykyaikaan ja ketterien menetelmien valtakauteen saakka. Tutkimuksessa esitetään konkreettisia ratkaisuja ohjelmistokehityksen tietoturvatyön tavoitteiden asettamiseen ja niiden saavuttamiseen. Tutkimuksessa myös tunnistetaan ongelmia ja haasteita tietoturvatyön ja ohjelmistokehityksen menetelmien yhdistämisessä, joiden ratkaisemiseksi tarjotaan toimintaohjeita ja -vaihtoehtoja.

Tutkimuksen perusteella iteratiivisen ja inkrementaalisen ohjelmistokehityksen käytäntöjen ja periaatteiden yhteensovittaminen tietoturvatyön toimintojen kanssa parantaa ohjelmistojen laatua ja tietoturvaa, alentaen täten kustannuksia koko ohjelmiston ylläpitoelinkaaren aikana. Ohjelmistokehitystyön automatisointi, työkaluihin pohjautuvat prosessit ja pyrkimys tehokkuuteen sekä korkeaan laatuun ovat suoraan yhtenevät tietoturvatyön menetelmien ja tavoitteiden kanssa. Tutkimuksessa tunnistettiin useita uusia tapoja yhdistää ohjelmistokehitys ja tietoturvatyö. Lisäksi on löydetty tapoja käyttää dokumentointiin, analyyseihin ja katselmointeihin perustuvaa tietoturvan todentamiseen tuotettavaa materiaalia osana ohjelmistojen suunnittelua ja laadunvarmistusta. Erillisinä nämä prosessit johtavat tilanteeseen, jossa tietoturvamateriaalia hyödynnetään pelkästään ohjelmistokehityksen ulkopuolisiin tarpeisiin. Tutkimustulokset hyödyttävät kaikkia sidosryhmiä ohjelmistojen kehittäjistä niiden tilaajiin ja loppukäyttäjiin.

Ohjelmistojen tietoturvatyö perustuu tietoon ja koulutukseen. Tieto puolestaan lisää kysyntää, joka luo tietoturvatyölle konkreettiset tavoitteet ja perustelut jo ohjelmistokehitysvaiheessa. Tietoturvatyön painopiste siirtyy torjunnasta ja vahinkojen korjauksesta kohti vahinkojen rakenteellista ehkäisyä. Kysyntä luo tarpeen myös uusille työkaluille, prosesseille ja tekniikoille, joilla lisätään tietoturvatyön tehokkuutta ja vaikuttavuutta. Tällä hetkellä kysyntää luovat lähinnä lisääntyneet tietoturvaa koskevat säädökset. Pääosa muutostarpeesta syntyy kuitenkin ohjelmistojen tilaajien ja käyttäjien vaatimuksista: ohjelmistojen tietoturvakyvykkyyden taloudellinen merkitys kasvaa. Tietoturvan tärkeys tulee korostumaan entisestään, lisäten tarvetta tietoturvatyölle ja tutkimukselle myös tulevaisuudessa.

Acknowledgements

Foremost, I would like to express my sincere gratitude to the supervisors, professor Ville Leppänen and assistant professor Sami Hyrynsalmi. It was your help, patience, and hard work that made this work possible. Thank you professor Pekka Abrahamsson for accepting the role of the opponent, and I also like to thank the reviewers, professors Alex Norta from the Technical University of Tallinn, and Martin Gilje Jaatun from SINTEF and the University of Stavanger. The work was carried out, and continues on, at the TUCS Software Engineering laboratory where I had the privilege to work with, gain advice from, and to follow the rigorous academic standards of Dr. Johannes Holvitie and Jukka Ruohonen, among others: This work would not have been possible without you. You are the greatest. I extend my gratitude to my colleagues and managers at CGI Suomi Oy. Thank you Jonna Lindholm for your rigorous quality control, cross-disciplinary feedback, and the countless inspiring discussions. Finally, special thanks to my extended family, especially to my mother. This work is dedicated to all of you.

In an attempt to summarize a dissertation project, much of what matters the most gets left out. At best, this has been a rewarding process of discovery and invention. Even at its worst, it has offered countless chances to restructure former priorities and preferences, even principles. The work continues.

Trondheim, 27 May 2019
Kalle Rindell

List of original publications

I Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2017a). Busting a Myth: Review of Agile Security Engineering Methods. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, pages 74:1–74:10, New York, NY, USA. ACM

II Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2018a). Aligning Security Objectives With Agile Software Development. In *In proceedings of XP '18 Companion, May 21–25, 2018, Porto, Portugal*, XP'18, pages 0–0, New York, NY, USA. ACM

III Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2015b). Securing Scrum for VAHTI. In Nummenmaa, J., Sievi-Korte, O., and Mäkinen, E., editors, *Proceedings of 14th Symposium on Programming Languages and Software Tools*, pages 236–250, Tampere, Finland. University of Tampere

IV Rindell, K., Ruohonen, J., and Hyrynsalmi, S. (2018c). Surveying Secure Software Development Practices in Finland. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, pages 6:1–6:7, New York, NY, USA. ACM

V Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2017b). Case Study of Agile Security Engineering: Building Identity Management for a Government Agency. *International Journal of Secure Software Engineering*, 8(8):43–57

VI Rindell, K., Holvitie, J., Ruohonen, J., Hyrynsalmi, S., and Leppänen, V. (2019a). Industry Survey of Secure Engineering Practices in Software Engineering. *Article in Review*

Authored and co-authored original publications not included in the thesis

- Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2015a). A comparison of security assurance support of agile software development methods. In *Proceedings of the 16th International Conference on Computer Systems and Technologies*, CompSysTech '15, pages 61–68, New York, NY, USA. ACM

- Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2016). Case Study of Agile Security Engineering: Building Identity Management for a Government Agency. In *Proceedings of Availability, Reliability and Security (ARES), 2016 11th International Conference*, pages 556–563

- Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2018b). Fitting Security into Agile Software Development. *International Journal of Systems and Software Security and Protection (IJSSSP)*, 9(1):47–70

- Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2019c). Security Assurance in Agile Software Development Methods: An Analysis of Scrum, XP and Kanban. In *Exploring Security in Software Architecture and Design*, chapter 3, pages 47–68. IGI Global

- Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2019b). Challenges in Agile Security Engineering: A Case Study. In *Exploring Security in Software Architecture and Design*, chapter 12, pages 287–312. IGI Global

- Rindell, K. and Holvitie, J. (2019). Security Risk Assessment and Management as Technical Debt. In *Proceedings of the IEEE Cyber Science 2019 Conference, June 3–4, 2019, Oxford, UK*. IEEE

Contents

Chapter 1

Introduction

APPLICATION
Those portions of a system, including portions of the operating system, that are not responsible for enforcing the system's security policy

Definition in military std. CSC-STD-003-85 (1985)

Software engineering is an extremely objective-oriented scientific and engineering discipline. It provides the tools and methodologies aiming to both defining the objectives of software, and the means to achieve them. In a software product, *security* is a key characteristic required to make software *dependable*, as defined in a taxonomy by Avizienis et al. (2004). In other words, secure software can be relied upon, also in adverse situations. As stated by Boehm (2006), software engineering is increasingly expanding to include practices traditionally labeled as system engineering. To address the security concerns in software development, security engineering is one of these system engineering practices software development processes will have to employ.

Software security has the goal of guaranteeing the confidentiality, integrity, and availability of software for its authorized users. In software development, this goal is achieved by the means of *software security engineering*: A set of processes and practices utilized to transform the defined *security objectives* into security features and functionality in the software. This is accompanied by producing appropriate *security assurance*, viable proof of security (Anderson, 2008). The desirable end result – "good" software – is a combination of solid software engineering process, completed with software security engineering practices, and executed to produce eco-

nomically and technically viable software, meeting its intended objectives. The term software security engineering reflects the aim to combine the practices of two engineering disciplines: security and software engineering. The aim of this discipline according to McGraw (2006) is the implementation of"secure features, not just security features".

This chapter opens with a definition for the term "application", taken from an old military computer security standard. The definition serves as an epitome of the traditional separation of duties and responsibilities in software security. It represents a culture where the requirements and functionality of the software itself are strictly separated from the requirements for security (see Yost (2015)). In this model, security was an afterthought and considered a hindrance to the efficiency of both the software and its development. In a typical approach, security controls were separate products used to monitor and restrict the software, installed after the software was already deployed in its operating environment. Software security was entirely dependent on the security of its environment and depended on external protection (McGraw, 2006). Security requirements, however, concern the software and the data directly. The dependency on external security (firewalls, security monitoring, etc.) can probably not be completely removed. The main benefit of software security engineering is the elimination or mitigation of the risk to business processes introduced by software security vulnerabilities: This can be best achieved by developing software with validated and verified built-in security.

In the time of ubiquitous networked personal computing, the importance of security in software products gained significant importance (Viega and McGraw, 2002). Despite this recognized requirement, mainstream software development methodologies remain security agnostic. The topic of security is approached indirectly at best, from much wider perspectives of quality assurance or architectural guidance (cf. Ambler and Lines, 2012a). In a typical industry setting, software engineering and security engineering still exist as detached practices. Security engineering is recognized as necessary to protect the software and the data, but the practices are performed either after the software engineering processes or parallel to it. This approach necessarily requires additional resources or time and fits poorly into the work methods of contemporary *agile* software development methodologies.

In software development, agility is characterized by lightweight processes and organization, used to control an iterative and incremental development process (Abrahamsson et al. (2002)). The agile methods were originally developed to meet the constraints of complexity, resources, and quality in software projects. In less than two decades they have gained near-ubiquitous mainstream dominance—largely due to their effectiveness. Security engineering now faces the challenge of adaptation to the efficient agile processes, and integrating into *agile software security engineering*. This transition can

be considered a primary prerequisite for effective production of secure software. The merger of these two fields of practice also acted as the primary motivator for the research described in this thesis.

The rest of this chapter is organized as follows: the background and motivation of the research are presented in Section 1.1; this is followed by the presentation of key concepts in Section 1.2. In Section 1.3 the research questions and the research objective are presented; Section 1.4 presents the research in this and related fields, and Section 1.5 gives the structure of this thesis.

1.1 Background and Motivation

Across the many definitions of software engineering a set of core themes recurs. Software engineering is *systematic work*, and in ISO/IEC/IEEE Standard for Systems and software engineering (2010) termed as *(systematic) application of knowledge or skill*, aiming to *produce quantifiable and working results*. Software engineering is also defined systematic method of producing software products for computers or other electronic devices, containing a set of key characteristics: the most important of these are functionality, reliability, usability, efficiency, maintainability and portability. From the practitioner's point of view, Bruegge and Dutoit (2003) approach software engineering through its various *rationale-driven* activities, resulting in *modeling* the real world into a software artifact by knowledge acquisition and problem-solving.

The definitions provided by e.g. Sommerville (2015) describe the characteristics of good software as *acceptable* or *fit for purpose*, *dependable and secure*, *efficient*, and *maintainable*. Addition of security to these characteristics is relatively recent, having taken place between 2005 and 2013. In comparison, software *safety*, a related concept, is still only implicitly included in these core definitions.

The characteristics attributed to good software by these definitions are integral to the purpose of the engineering process and the products resulting from it. As an emergent and applied science, software engineering is related to computer science and system engineering; these are in turn emergent of their relative scientific fields. The core principles of software engineering can be epistemically and ontologically reduced to their roots in natural and social sciences. Its independence is justified by the emergent properties containing features that do not exist in any of the root properties. A core set of properties attributed to a desirable software product, produced by a feat of software engineering, contains properties such as *robustness*, *reliability*, *correctness*, and *effectiveness*. When these are implemented in a proper manner, a good basis for *security*, along with other properties, is achieved.

In this sense, software security engineering, as protection from unauthorized or unwanted use of software and the data it stores or processes, is further emergent from software engineering (cf. Avizienis et al. (2004); Sommerville (2015)).

Information system security and the related *security assurance* are typically based on the concept of "defense in depth". This security model is promoted by key security authorities, such as ICS-CERT (2016)[1]. In this approach, security is built in layers consisting of multiple security controls surrounding the protected information assets, described by e.g. Cleghorn (2013). Maintaining a credible and effective *security response* requires skill and resources, yet the amount of reported security breaches keeps growing steadily according to the Common Vulnerabilities and Exposures (CVE) tracking in the United States National Vulnerability Database (NIST NVD (2018)) and globally to MITRE CVE (2018). Despite the coordinated global effort, the economic and intellectual property losses resulting from the security incidents are on the constant rise according to industry reports by e.g. Ponemon Institute (2017, 2018). Interestingly, the reports also show a shift in the industry's security spending from network security towards application and data security. The effects of this work will be evident only in the long term, but the trend indicates a call for increased research in software security engineering.

In the history of software engineering, standardization has been a principal driving force of security. The current ISO/IEC, IEEE, and most national standards can directly trace their origins to the 1970s-era United States Department of Defence's standards and federal procurement guidelines. Later, these are merged into a number of other national and international standards to form the current international software standard set. The security standards are even more extensively based on U.S. military and other federal government specifications, according to historic recapitulations by Yost (2007, 2015) and Bayuk and Mostashari (2013). The standardization work started in the late 1960s—around the time when software engineering had begun to be recognized as a discipline separate from systems engineering, and computer science began to gain its status as an independent branch of science. Besides providing regulations, the main purpose of standardization in engineering is to maintain and share the best practices in their representative area of expertise.

Upon their introduction, the agile software engineering methods were criticized for having a detrimental effect on software security and lacking characteristics necessary to comply with formal security models. These concerns and criticisms were brought forth by e.g. Conboy et al. (2005). Initial research was committed to applying security to the agile methods by e.g.

[1]The Industrial Control Systems Cyber Emergency Response Team

Wäyrynen et al. (2004), basically concentrating on the feasibility of using agile methods to produce secure software as defined by e.g. compliance requirements. According to a thorough empiric research in the agile security requirement elicitation by Ramesh et al. (2010), problems appear to rise from the *prioritization* of the requirements. However, requirement management and prioritization are among the key principles of the agile methods (see e.g. Schwaber (1995); Beck (2000)).

Traditional security engineering processes rely on a deterministic, process-based and sequential approach. This approach aims to minimize the security risk by introducing more or less rigid processes to meet the various acceptance criteria, validated at pre-fixed milestones. From the security engineering point of view, agile methods are seen too feature-focused to allow for the necessary security analysis or planning. Agile development has also been seen by e.g. Kruchten (2010) to have negative effects on architectural planning, including security architecture; this view is further discussed and partially reconciled by Abrahamsson et al. (2010). As the security requirements are often categorized non-functional, following the agile principles is considered likely to cause security requirements to be underprioritized as unnecessary overhead. Despite the ongoing research effort, security practitioners still encounter difficulties in applying security engineering to agile projects, as observed by e.g. Türpe and Poller (2017).

1.2 Key Concepts

This chapter provides the terminology and definitions for software engineering and security engineering. Key research items are presented and the research challenges positioned within their respective fields.

1.2.1 Software Engineering in Security Context

During its relatively short lifespan, software engineering has gone through some comprehensive transformations (see Sommerville (2015)). The evolutionary process of the software engineering methodologies reflects the fundamental transformations undergone by the software industry – and the fields of its applications. The development in the fields of electronics, networking, and software tools during the past few decades has been unprecedented, as are the skill requirements to be able to use and develop them. These changes reflect also to the methodological level, although with a delay in wider adaptation. In software development, however, the traditional management approaches have been effectively replaced by agile methodologies as shown by VersionOne (2018) and already noted by Abrahamsson et al. (2002).

Prevalent use of the adaptive, iterative, incremental and lightweight methods contrasts the classic security engineering methodologies. The increasing requirement for software security will require introduction of security engineering techniques into software development. Software security is more than a subset of software quality assurance. It introduces security risk and requirement assessment techniques to form security policies, security engineering techniques to implement the policies, and the security assurance component to provide evidence of successfully met security objectives. Software engineering and security engineering share the goal of robust, dependable, reliable and economically feasible software. Managing the security requirements and processes should not diverge from the management of software development processes. To maintain effectiveness under strict security requirements, convergence is necessary to perform at the methodological level.

1.2.2 Security Engineering and Software Security Engineering

A conceptual definition of security engineering by Anderson (2008) states it is "*the practice of building systems to remain dependable in the face of malice, error, or mischance*". The practice of *software security engineering* forms a distinct field within both software engineering and security engineering. It exists to mitigate man-made risks and threats to software-intensive systems. It introduces a wide set of constraints and requirements to the development process, typically requiring a distinct set of development-time activities to be performed in various stages of the software development process. Software security engineering practices and activities are integrated, or at least directly related to the software development practices (see Viega and McGraw (2002)).

Sommerville (2015) divides information security into three domains: *infrastructure security*, *application security* and *organizational security*. Software engineering primarily concerns applications; software security engineering may be thus defined as security engineering performed during the software development phase of the software life cycle. In order to be effective, elements from all three domains are required.

To further specify the structure of software security engineering, the Systems Security Engineering Capability Maturity Model (SSE-CMM) defines software security engineering to consist of three security processes for the management of security risk, security engineering, and security assurance. In the development of secure software, these can be considered the key areas to be addressed by security engineering frameworks, or security development lifecycle models. The SSE-CMM is standardized as ISO/IEC Standard 21827:2008 (2008). To complete this structure, also the context in

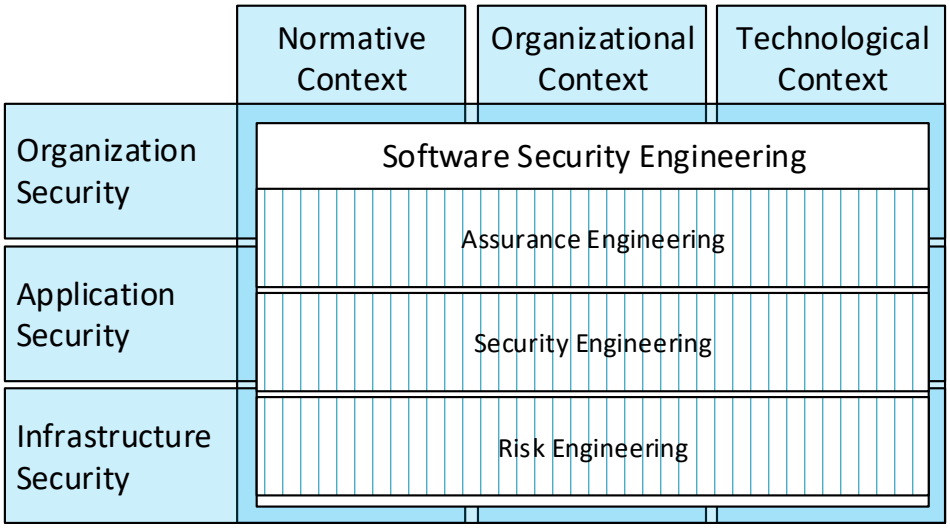


Figure 1.1: Software Security Engineering in standard-derived Information Security domains and contexts, as derived from ISO/IEC software security standards

which security engineering is performed needs to be defined.

The capability frameworks generally follow with the traditional Plan-Do-Check-Act (PDCA) cycle. The PDCA is an iterative management principle, tracing its origin back to the statistical management methods by Shewhart and Deming (1939). Continual improvement, a central theme in the agile methods, provides a mechanism to prevent the organizational standards from becoming "dogmatic" by continually maintaining their relevance. It also helps to maintain a relevant *security rationale*, justification of the security effort.

The ISO standard for application security, ISO/IEC Sstandard 27034-1:2011 (2011), uses the notion of three security contexts: *normative*, containing laws, standards, and other regulation; *organizational*, containing people, policies, and business processes; and *technical*, containing the technological assets.

These activity areas and security contexts form the basis of how security engineering is examined in this thesis. The conceptual framework is presented in Figure 1.1.

The diagram is made from the viewpoint of software security, and represents the relationship between the key concepts. Software security engineering concerns each of the three domains, and each of the three defined contexts. Software security engineering contains the three main processes: risk, security and assurance as defined by the SSE-CMM.

The primary motivation of the research presented in this thesis is to

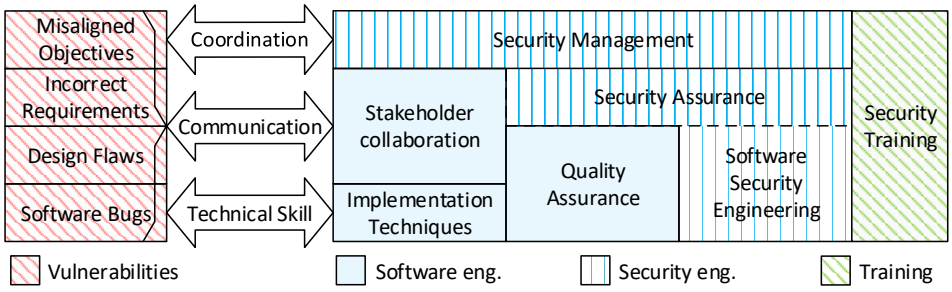


Figure 1.2: Security vulnerabilities in software development ways to address them by software security engineering

examine how software security engineering is to be conducted in agile settings, and how both security engineering activities and software engineering methods should be adjusted to further guarantee quantifiable, working, and *secure* results. In the software industry, security engineering is often practiced by following open and public precompiled guides, security frameworks and security development lifecycle models. In addition to the organizational and technological constraints, the security work is guided by software security standards, and other regulation (cf. Schneider and Berenbach (2013)).

Software security engineering combines the practices of both software engineering and security engineering and extends them. It deals with security vulnerabilities specific to software applications introduced during their design and development. An overview of this field is presented in Figure 1.2.

The main causes for security vulnerabilities are placed on the left, and the methods are presented on the right, supported by organizational security training. Arrows depict the concepts of how the vulnerabilities are addressed in a software development process. The diagram also depicts the distinction between the mechanisms of software engineering and security engineering and their limitations. For example quality assurance may not adequately verify and validate the security requirements, and security management is needed to correctly coordinate the security objectives and to create a manageable security rationale. The framework in the diagram is derived from the body of security standards and models presented in the following chapters, applied to software development.

Although qualitative requirements like robustness, reliability, and correctness are inherent to secure software, software security is not achieved by quality improvement alone (Howard and Lipner (2006)). To meet the properly set security objectives, a definitive set of security engineering activities is also required. By performing a set of tests, the absence of software security can be verified. Poor quality, such as design flaws and implementation bugs are a security threat even without malicious intent. Another clear distinction between traditional software engineering and Quality Assurance (QA)

efforts is seen on the requirement level, where the verification and validation of security-specific requirements typically require security-specific testing. Validation, in this context, refers to the relevance of the tests themselves: they link the security tests to the security objectives. The produced *security assurance* is employed as evidence of how the security objectives have been met, and the risk management controls implemented.

Security assurance forms a significant part of the results of security engineering. It is used to provide evidence of the security implementation, to manage security risks, and to evaluate the security of the software. Assurance itself is produced through various assurance techniques: reviews, interviews, analyses, and security testing. Security validation can also be performed by an independent third party, performing security audits in the form of witnessed tests or public reviews (Such et al. (2016)).

The foundation of software security engineering is training and skills. The right skill set, combined with the right tools, is the best remedy for implementation-level issues. Security training is a ubiquitous requirement in security work and may form a significant part of the security assurance. Skill allows the detection, avoidance and even anticipation of software bugs, design flaws, or incorrect or missing security requirements. Clear communication with the stakeholders is another direct mechanism to reduce all types of security vulnerabilities. A high-level coordination effort is also typically necessary to maintain the security rationale through security objectives.

Security engineering in a software-intensive system is a composite of the information security concepts presented above, and discussed further in the following chapters. Research in this thesis concentrates on the security engineering activities applicable to software engineering. These activities are primarily applicable during the software development lifecycle (see ISO/IEC Standard 15288 (2015) and ISO/IEC Standard 21827:2008 (2008)), and aim to fulfill the security requirements of the software product under development. Furthermore, the focus is on the methodological level: How software security engineering can be integrated into the modern agile and lean development methods, used to build diverse and complex software-intensive systems.

1.3 Research Questions

The research is motivated and was initiated by the essentially important requirement of software security. To achieve this goal and to set the scope for the work, and to ensure the usefulness of the results, the following preconditions were set:

- Industry standard software development practices, i.e., lightweight iterative and incremental methods are applied (agile development);

- The mechanisms to implement software security are defined by industry standard software security development life cycle processes and maturity models.

In software security engineering, the security rationale, which can be understood as a set of security objectives, is used to set the security requirements. Consequently, this research focuses on the software security engineering practices used to achieve these objectives by correctly eliciting the security requirements, ensuring the correctness of the security engineering implementation, and the effectiveness of security assurance. This framework helps to formalize the research objective, development and construction of secure software products: *How can secure software be produced using agile software development processes*? The research questions are derived from the research objective, and become more specific as the security methodologies and mechanisms are taken under scrutiny.

Part of the research objective is finding efficient and effective ways to fulfill the security rationale, in conformance with existing software development practices. The research focuses on the methods of software security engineering, the practical activities involved in making software secure and producing security assurance. Validating the use of these activities included conceptual research, a case study, and surveying the industry practices in software development and security engineering.

RQ1: How are software security objectives and practices aligned with agile software development?
This research question concentrates on the definition of security, and the security objectives, targets and requirements set in the software development process. Software engineering is, by definition, translation of requirements into functionality. Security requirements undergo the same process, but defining the security requirements and prioritizing their implementation is a challenge for software development organizations with conflicting objectives and limited resources. Understanding the security objectives and correctly defining security requirements is the basis of the software security engineering work. This research question is answered on the methodological level by defining a concrete cross-section of software security standards, security policies and the comparable agile software development practices related to objective setting and requirement definition.

RQ2: How can software security engineering be integrated into agile software development methodologies?
Security engineering during software development is performed by a set of activities and processes, selected to meet the security engineering objectives and to provide sufficient security assurance. This research question is answered by examining the existing security maturity and security development life cycle models both in relation to the software development

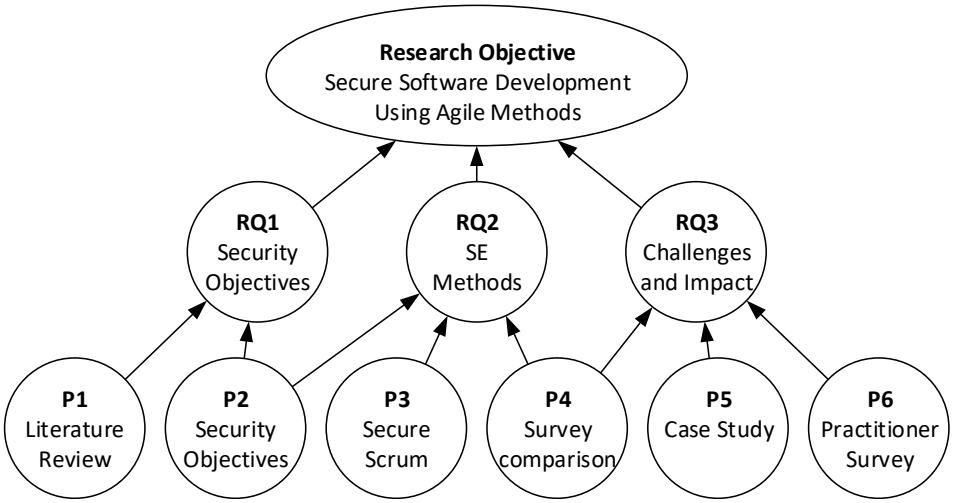


Figure 1.3: Research objective, research questions and the related publications

methodologies, concrete software security development work, and the security objectives examined in RQ1. Specific attention is paid to fitting security engineering tasks into an iterative and incremental development process, done in a way that conforms with the agile and lean methods and conforms with the agile software development practices. The research focus is in choosing the appropriate security engineering activities and combining them into the agile software engineering processes and tools. The goal is to enable the software development organization to achieve the security objectives in an effective manner and in better coordination with other, potentially conflicting objectives.

RQ3: How is security engineering affecting the agile development process and the security of the software?
This research question is aimed to identify the challenges and security impact of integrating software security engineering into iterative workflow – the themes central to RQ1 and RQ2. Compliance with security standards and organizational security policies, and the resulting requirements for security assurance are considered an important factor in software security engineering. These security engineering practices and their consequent effect on the actual improvement to the security of the software products are items of specific research interest.

The research structure and the publications providing the main contribution are visualized in Figure 1.3. The publications are described in further detail in Chapter 4.

The research objective is to define and develop methods and techniques for producing such software that meets its security objectives. Additionally,

this objective has the condition of utilizing the agile methodologies currently prevalent in the software industry. In the given descriptions, the adjective "agile" is used to denote the software methodologies currently prevalent in the software industry. The practices and activities characteristic to these methods are presented in Chapter 3.

1.4 Related Research

Security engineering is an active area of research with multiple subtopics. Agile software engineering has become a subject of research in early 2000s, with early work conducted by Wäyrynen et al. (2004), Kongsli (2006), and Boström et al. (2006). While much of the early research interest concentrated on the feasibility of agile software security engineering in general, Ge et al. (2007) made one of the initial efforts to develop and describe agile architectures. Agile quality mechanisms were examined by Huo et al. (2004). More conceptual agile software engineering research was conducted by Baca and Carlsson (2011), with empirical experimentation by e.g. Ayalew et al. (2013), Oyetoyan et al. (2016), and with some noted challenges, by Türpe and Poller (2017). Security assurance in agile development has been focused on by Beznosov and Kruchten (2004) and ben Othmane et al. (2014). A similar objective to this research has been chosen also by Stirbu and Mikkonen (2018), aiming to achieve regulatory compliance more efficiently by utilizing agile methods, concentrating on the field of software safety.

From a software engineer's point of view, software security engineering requires an understanding of the security objectives (what and why), and awareness of a wide array of security techniques to implement the required security features and functionalities. Requirements engineering, risk management, quality improvement, estimates and metrics, and formal methods are among the closest practices required to further develop software security engineering. The improvement process will start at security awareness, created by security research and training. Security is a relatively new focus area in software engineering, and increasing regulatory security constraints force the industry – and research – to meet the demand in an economical and efficient way, inherent to prevalent software engineering methods. The future of security engineering is shared with software engineering, and this requires confluence and compatibility with agile methods.

The reduction of implementation-time errors is a central research topic. In this work, security awareness was identified to be a crucial factor in the systematic reduction of security vulnerabilities. A solution suggested by Adelyar and Norta (2016) is agile agent-oriented modeling, bringing the security principles a concrete part of the software design and implementation. Another approach to find a practical way to improve software security is

aspect-oriented security programming and design, a concept suggested by De Win et al. (2002) and extended upon by e.g. Boström et al. (2005). In industry, the reduction of implementation is typically performed by following a precompiled list of instructions – a concept described by Tsipenyuk et al. (2005).

A primary mechanism to provide security assurance is to perform various forms of verification and validation. Recent work in the validation and verification, and the involved methods is extensively summarized by a systematic literature studby by Such et al. (2016) An analysis of risk-based testing approaches was performed by Felderer and Schieferdecker (2014). Current issues and challenges in security testing in software development have been reported by Cruzes et al. (2017).

Software safety is a closely related field of research: Software security and software safety share much of the same methodology and rationale. Agile methods have been used for safety critical work by Fitzgerald et al. (2013); the effect of safety regulation to DevOps has been considered by Laukkarinen et al. (2018). The agile development process has been adopted to formally comply with Capability Maturity Model's maturity levels by e.g. Marcal et al. (2007); Jakobsen and Sutherland (2009). Cases where hindrances in applying agile to security engineering were found, or agile being detrimental to software security have also been reported by e.g. Türpe and Poller (2017). This indicates a requirement for further research in agile security engineering, and that a divergence in the ways to efficiently produce safer software is required. Research concerning software standards regarding requirement engineering has been extensively mapped by Schneider and Berenbach (2013).

Security engineering and software engineering are cross-disciplinary practices. *System security engineering* is defined in the United States Department of Defence's guide for protection of trusted systems and networks (2017), page 14, as "*an element of system engineering that applies scientific and engineering principles to identify security vulnerabilities and minimize or contain risks associated with these vulnerabilities*". Information security research draws methods and theories from e.g. philosophy, history, political science, sociology, psychology, law, statistics, computer science, physics and mathematics, as stated by Anderson (2008).

Software security engineering specifically concentrates on the security issues within software engineering: Security metrics, formal software development methods, programming languages, software development methodologies, and security techniques and tools. There is a substantial overlap of methodologies and practices with software security and software quality improvement, as quality assurance techniques and tools alone do not sufficiently address the security-related requirements. Along with most fields of computing, software security engineering is also an opportune field for applications of machine learning and artificial intelligence research. Behavioural

sciences, such as industrial and organizational psychology, are directly applicable to the design and development of secure information systems.

In addition to the above fields of research and application, the research topic includes several directly security-related fields of application. These include protection of intellectual property, identity management, cryptography, communication security, privacy, investigation of computer crime, and information warfare. Increasing legislative and regulative pressure for protection of privacy and related immaterial rights promotes the privacy-related topics, mandating strict privacy policies for the management of sensitive data. Privacy can not exist without security.

Software security engineering necessarily shares methodologies and characteristics with mainstream software engineering methods (Viega and McGraw (2002); McGraw (2006); Anderson (2008)). Consequently, following the software engineering trends, software security engineering is predominantly based on automated tools. Software security is also increasingly formalized by regulative requirements, mandating not only secure coding practices, but concrete security features and functionality as a requirement for software development. The methodologies are presented in a design science framework, as presented by Hevner and March (2004).

1.5 Structure of The Thesis

The thesis consists of an introduction part, divided into four further sections, and the main research contribution given in separate but connected research articles. The following sections present the field of software security engineering. Section 2 presents the knowledge base for the security development process, by presenting the core international standards and techniques of software security engineering and their history in an ethnographic framework. The central ISO/IEC software and application security standards are presented. The standards are complemented by presenting the central industry- and community-sourced security development and maturity models.

The security engineering standards and methodologies are used to form an analytic framework, against which the selected agile software development methods are presented. The agile methods most commonly used for the build process are presented in Section 3. An ethnographic retrospective of the methods is presented, and a security engineering analysis performed along with the guidelines presented by Hevner and March (2004). The structural components of each method and the benefits and challenges experienced in applying them into software security engineering work are drawn from extant literature. Specifically, the analytic focus is in the usefulness and effectiveness of individual software development practices for

security engineering work.

In Section 4, the individual peer reviewed publications are presented, and their contribution summarized. Section 5 presents the main findings and their implications and concludes the thesis by providing directions for future research in software security engineering.

Chapter 2

Software Security

In software engineering research, software security is considered an essential part of the *correctness* of a computer system or communications, denoted by the concept of *dependability*. A definition of dependability states it to be "a generic concept including a special case such attributes as reliability, availability, safety, confidentiality, integrity, maintainability, etc." as defined by Avizienis et al. (2004). Software security, as a part of the broader field of information security, is in ISO/IEC Standard 15026-1:2013 (2013) defined as the practice aiming for *protection of system items from accidental or malicious access, use, modification, destruction, or disclosure*. This also effectively comprises the attributes of confidentiality, integrity, and availability.

Software security engineering activities are specific to the protection of computer and software data, assets, and processes. The key concepts and attributes of information security are depicted in Figure 2.1 from the software security perspective.

This classic diagram visualizes the information security elements, applied to the context of software security. The triangle represents the interdependent core attributes forming the security objectives: *confidentiality*, *integrity*, and *availability*. In this depiction, the sides form the system boundaries, and contain the protected asset, such as a service or data. The software resides in an operating environment, which may host other connected systems, and connect to other potentially unknown system environments. Operating environments consist of hardware, networking, software, and people in various roles. Typically this is run by an organization, and guided by policies, regulations, and laws (cf. ISO/IEC Sstandard 27034-1:2011 (2011)).

Organizations achieve security objectives by implicit or explicit security policies. Policies are enforced by various security mechanisms, implemented into the operating environment and, during the software development pro-

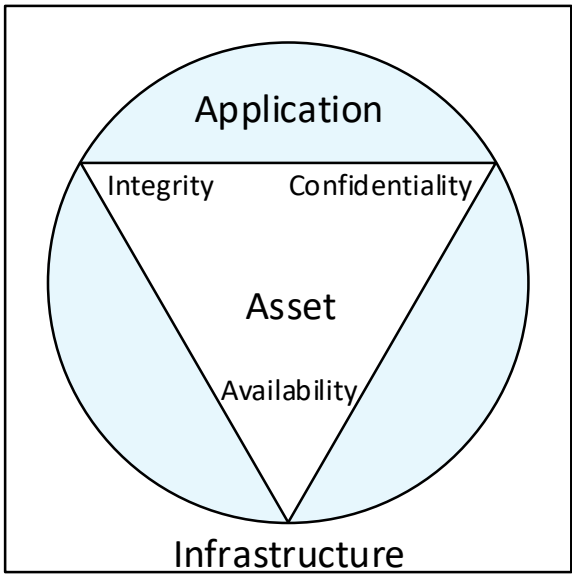


Figure 2.1: Software security elements and characteristics

cess, the software itself. These mechanisms include authorization, restricting access to the protected assets and authentication, verifying the authorization. Various cryptography methods are used to verify the authenticity of the accessing parties, as well as to encrypt the data and the other software-protected assets. Data may be in various states, such as in storage, being processed, or being transferred between system entities.

Encryption of the data and the communications is a cornerstone of security. It reduces the risk of unauthorized use of data after the other methods have failed. Encryption does not, however, protect against availability risks: combined measures taking place in the software and operating environment are required to ensure availability. Encryption is also an effective means to reduce the security risk, although an extensive encryption framework may be costly to implement and maintain (Ponemon Institute (2018, 2017)).

Besides governing software development, the security policy framework covers the whole software life cycle, extending also to other domains beside application security: infrastructure and the organization. This includes physical security of the computing assets, security controls for the personnel processing, accessing and operating the assets. Security policies can be further divided to corporate, organizational or technical, as specified by Baskerville and Siponen (2002), an approach quite close to the ISO standard for application security, ISO/IEC Sstandard 27034-1:2011 (2011).

Security engineering involves a wide range of methods and techniques. Besides computer science, the disciplines vary from cryptography to sociology, law and psychology (Anderson (2008)). The means to breach informa-

tion security stem from the same source as the means to protect it. The adversaries typically have the benefit of drastically less limited resources: most importantly, they have considerably less limiting time constraints, and the benefit of accumulated post hoc vulnerability information.

In a software lifecycle model, in which the software is first developed and released, and then potentially gets updated in the maintenance phase, the long-term focus has traditionally been on operational security. A shift towards iterative development has introduced various continuous delivery models. These models enable software developers to implement and deploy security engineering response in reaction to changing or entirely new security requirements. In addition, new security threat models put an emphasis on the security of the software itself, not just the environment.

Figure 2.2 depicts the lifecycle of an advanced security threat: insertion into an information system, exploiting security vulnerabilities, and achieving the attacker's objectives.

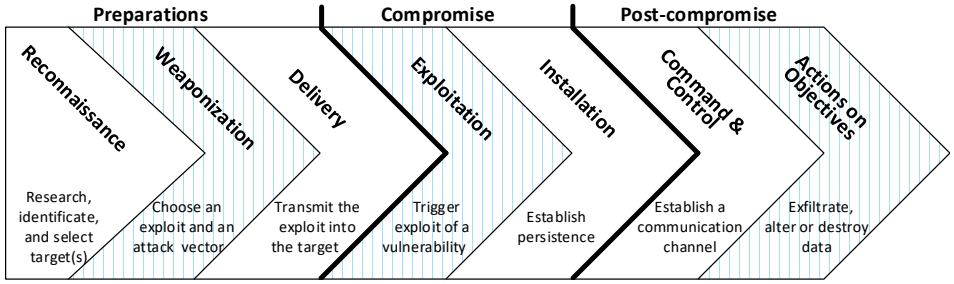


FIGURE 2.2: Lifecycle of an advanced software security threat (adapted from Hutchins et al. (2011))

The diagram helps illustrate the progressive phases during a security exploit. Although the threat must not go through all the individual phases, security *forensics* indicates that exploits and threats may reside within the operating environment for prolonged periods of time. After the firewalls and other external protection measures have been bypassed, software vulnerabilities are probed for, and possibly exploited, for extended periods. The total length of a security breach may be several months before an exploit is utilized. The exploitation continues until it is detected, or the adversary decides to terminate the activity. In the latter case, the incident may never be detected (Krutz and Vines (2010)).

Protecting the system boundaries and the operating environment is a concerted effort, requiring an organization to employ multiple security engineering practices. Security is a combination of security engineering efforts, and should be prepared to detect and prevent security threats and vulnerabilities also in each application system. Software security engineering acts to improve the security at all the stages of an incident, as presented in Figure

2.2, by the following mechanisms:

- reducing the attack vectors,
- reducing the number of available exploits, and
- reducing the ability to act on the objectives.

In software development, security objectives are set also by legislation and regulation. The increased demand for security assurance calls for additional techniques and processes incorporated into the software development methodologies. These techniques and security procedures may be costly and time-consuming to implement. Especially in software development, it can be very challenging to attain formal security maturity levels in an economically feasible manner, as reviewed by Silva et al. (2015).

The drive for secure software requires a new combination of cutting edge software engineering methods. These need to be able to flexibly meet the various requirements of the system development, and ready to accommodate the core security processes and practices required to achieve the security rationale. This combination would not only address the security requirements but also benefit the whole software development process and software lifecycle, resulting in reduced risk and lowered lifetime cost (Howard and Lipner (2006)).

2.1 Software Security Standards

Early information security, with central computers in access-controlled locations, was primarily about physical security and personnel control. Cryptography was primarily a means to secure communications, both electronic and otherwise, with fewer considerations for encryption of stored data. First formal requirements for computer security were made by the US Department of Defense (DoD) in 1967 "to address computer security safeguards that would protect classified information in remote-access, resource-sharing computer systems". This specification can be found in the history section of one of the first comprehensive information security standards, the Orange Book (DoD (1985b)). The book is called such simply by the color of its cover. Together with a supplementing standard presenting implementation guidance (DoD (1985a)), presented formal evaluation criteria and simplistic selection matrix for security policies to be enforced, based on the strict classifications of data and the personnel accessing it.

The late 1960s was a crucial transition period to both software and security engineering: batch-based computing began to be replaced by time-sharing of mainframe resources and networking, posing the computer operators and programmers with unprecedented complexity and a new array of

concepts. The users gaining remote access to computer's internal data and processes through connected terminals created a new set of programming challenges, and also a requirement for security. Drastic quality improvement efforts were necessary to prevent the users from crashing the computers and corrupting data by issuing malformed programs, commands or data. The era and its problems, known as the "software crisis", is discussed in more detail by e.g. Dijkstra (1972). This is also the time when the standardization work in both software and security started, addressing an increased requirement for secure high-quality software. The root of both can be traced into the United States' federal government guidelines, especially those of the Department of Defense.

Military specifications, such as the Orange Book, are purpose-built for strict hierarchies and have required a thorough "demilitarization" to be useful in a wider array of organizations. The root source of the current international computer standards is in these early US and NATO military standards. For wider applicability, they are combined with a number of issues by U.S. National Bureau of Standards (NBS, now NIST), and publications by other national standardization institutes, most importantly British Standards Institution (BSI) in the United Kingdom, and Deutsches Institut für Normung (DIN) in Germany. The main body of these is maintained by the International Standardization Organization (ISO), more specifically the International Electrotechnical Commission (IEC). The committee developing the information and ICT protection standards (Joint Technical Committee 1, Subcommittee 27) is currently chaired by DIN. The ISO/IEC standardization work is increasingly combined with the work from the US-based Institute of Electrical and Electronics Engineers (IEEE).

The ISO standards form only the general core of the standardization body, completed with extensive national and industry-specific regulations. These form complex interdependent frameworks that may be very difficult to follow and therefore benefit from. An especially complicated example of such a framework is the current U.S. DoD cybersecurity policy and guidance matrix: the extensive amount of documentation included in this framework is shown in Figure 2.3.

The chart itself is here presented merely as an example of a well-matured and extensive normative framework with a downside of complexity. It contains 195 linked documents in 16 functional categories. The documents range from national cybersecurity strategies to operational instructions of the specific technologies. While this may seem complex, the United States federal system security framework, Ross et al. (2014), contains a numerically even more impressive effort with 30 processes, 111 activities and 428 defined security engineering tasks. This work, published in November 2016, provides an example of an elaborate security scheme with heavy emphasis on security assurance. The regulative pressure on software development is

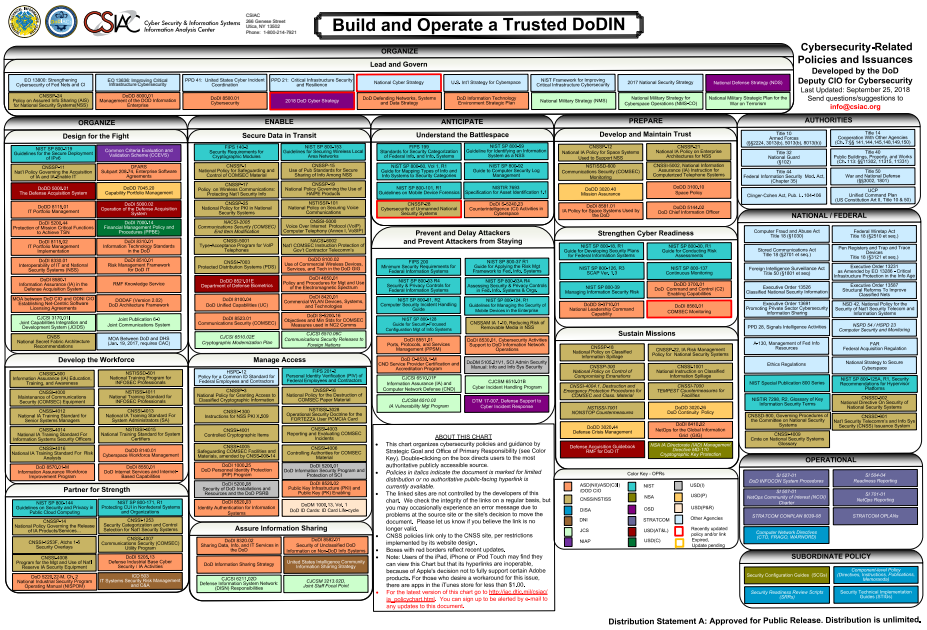


Figure 2.3: Example of a complex and extensive organizational normative framework by the Department of Defence Information Analysis Center (2018)

propagated by regulating the procurement process, setting guidelines and assurance requirements. It should also be noted that these frameworks are likely to contain only unclassified documentation with additional documentation withheld from the public.

In Finland, the government initiative for cybersecurity, VAHTI, provides instructions for security implementation (VAHTI (2016)). It helps instantiating the guidelines of Finland's current national cyber security strategy (Rantala and Kievari (2016)). VAHTI currently consists of 49 standing instruction documents. A single one of these instructions, the VAHTI 1/2013 (2013), sets the regulatory requirements for the software development processes. This in the form of a three-level maturity model, depending on the security classification of the protected asset. In wider industry standardization use, the term "cybersecurity" is currently used mainly in the context of industrial automation, regulated by ISO/IEC Standard 62443 series.

To maintain a clarity of the key concepts and the number of standards manageable, only the central ISO/IEC standards concerning software security are under scrutiny. The scope is limited to software security engineering. The standards are used to provide key terminology and the main concepts of security engineering used throughout the later sections. Dependencies and

positions of the most important security standards in the ISO/IEC software standardization are presented in Figure 2.4

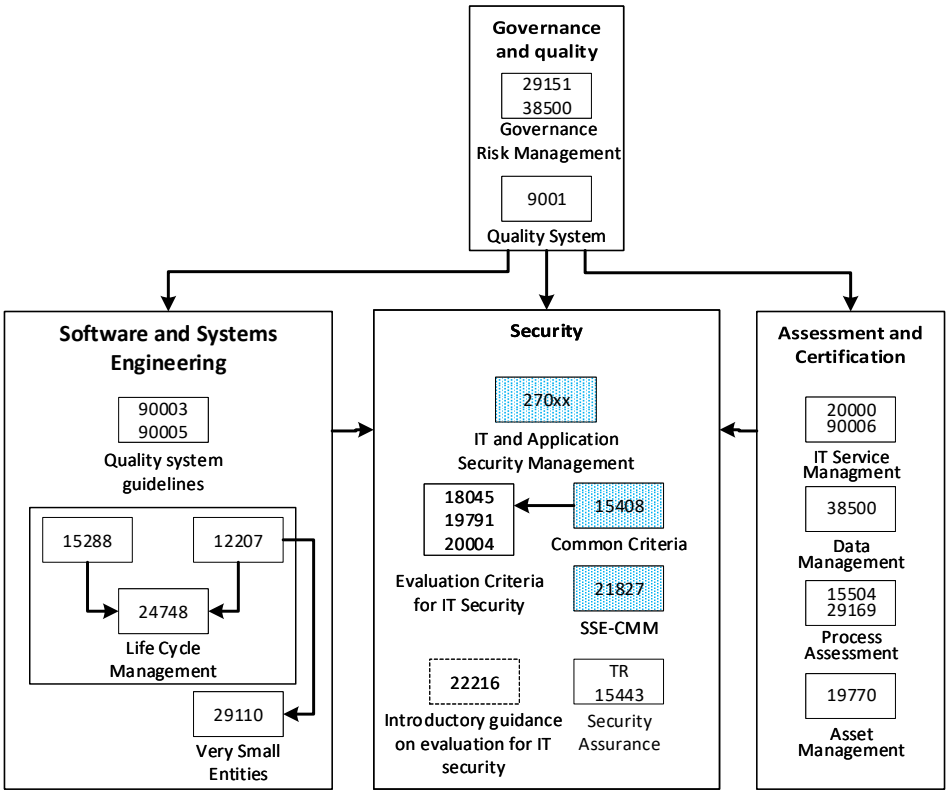


Figure 2.4: Related ISO/IEC software and security standard structure

The standards presented in Figure 2.4 are organized into four groups: governance and quality standards; the systems and software engineering lifecycle management standards within the former; the assessment and certification standards; and the standards for assessment and implementation of security. The focus is on the security standards, presented by the colored items in the center of Figure 2.4. The adjoining standard categories provide the necessary points of reference and connections for general governance, quality, and other standards regarding software engineering.

To clarify the concept of software security, three core ISO/IEC security standards directly involved with software development are examined closer. The first of these is the Common Criteria ISO/IEC Standard 15408-1:2009 (2014), which provides security guidance for evaluation of security of the software products and their operating environment. The second one is the SSE-CMM ISO/IEC Standard 21827:2008 (2008), which provides a semi-formal process framework for security engineering. The Common Criteria is used to derive the definitions and requirements for security assurance, and

concrete criteria for evaluation of security objectives. From the SSE-CMM, a set of security engineering and risk management processes is presented, and a basic security engineering framework is extracted. These two main standards are set into a wider perspective of information security and application security management by the standard for application security, ISO/IEC Sstandard 27034-1:2011 (2011). Much of the terminology and principles in this research work are derived from these, cross-referenced with related standards. It should be noted that there are separate standards also for security assurance (ISO/IEC Standard 15026-1:2013 (2013) and ISO/IEC Standard 15443-1:2012 (2012), and a risk management framework (ISO/IEC Standard 27005:2018 (2018)). The ISO technical report 22216, currently still under development, is being created to make it easier to apply the Common Criteria.

For each method or security framework presented, applying it to agile development methods is discussed. The bulk of ISO standards predate the agile methods. Even after recent updates they still maintain to concern the ISO software life cycle models only, such as the ISO/IEC/IEEE Standard for Systems and Software Engineering Life Cycle Management (2018). Adapting these models and methods for agile software development can be very challenging, and there is much room for improvement. In this adaptation process, the main goal is to maintain flexibility to provide a viable framework for a wide variety of security objectives.

2.1.1 Application Security

The ISO/IEC standard series for application security, the ISO/IEC 27034, is an exhaustive yet complex work on the aspects of security in software applications. The scope of these standards is to define the security of software applications throughout their life cycle, complementing the more development-oriented Common Criteria and SSE-CMM. The series currently consists of seven main parts. Instead of techniques, this standard series defines the governance of application security at various phases of the life cycle. It contains definitions, security frameworks, terms and concepts, and clarifies the role of security engineering in various phases of the software application's life cycle. Published over the course of several years, later parts define the application security management and application security validation processes, and application security data controls.

The central concepts in this standard are *Organization Normative Framework* (ONF), the *Application Security Management Process* (ASMP), and the *Application Security Controls* (ASC). In this formalization, the ONF for the ASMP is provided by separate organizational management processes, resulting in *Application Normative Framework* (ANF), forming the security requirements for the application. The approach to security improvement

is strictly based on risk management, and the aim is to provide means to evaluate and mitigate the security risks. ONF is specified in 27034-2 and the ASMP in 27034-3.

The security framework is built on the principle that the organization, utilizing a risk management process, comes up with a *Targeted Level of Trust* for an application; the application's *Actual Level of Trust* is then validated using an elaborate application security scheme framework, specified in 27034-4. The validation of the achieved level of trust is achieved by a combination of security testing and other means to provide security assurance. Formal security audits are heavily utilized to ensure formal compliance with the standard.

This standard provides the governance model and principles for application security management. Besides a recent extension describing "Case Studies" (27034-6), the standard does not specify the security engineering activities themselves, nor does it provide concrete criteria for their evaluation. Instead, it describes the security-related processes involved in the procurement and operating of software. The actual development-time software security engineering activities are presented in the related standards, presented in the following sections. These basic concepts derived from ISO 27034 series form also the analytic framework on software security engineering used throughout the following sections.

As stated, the origins of the approach reflected in the Common Criteria are in the Orange Book, from an old military computer security standard series. The evaluation framework was defined in the DoD security standard 5200.28-STD or CSC-STD-001-85, "Trusted Computer System Evaluation Criteria". A directly related standard, CSC-STD-003-85, set the criteria for different classes of Trusted Computing Base (TCB), which included the software running on these platforms. An essential role of this accompanying standard was to provide a mechanism to evaluate the security requirement based on the preset security levels of the data and the users accessing the system.

The original approach suited the limited amount of computers, systems, and users of the time. This, combined with the context of strictly hierarchical military organization, enabled the use of a simplistic instructional guidance matrix to determine the security level for any computer system. In this guidance, first the security target was defined based on a predetermined classification of the data contained in the system. Based on this, the standard dictated the requirement criteria for evaluation, including the security clearance requirement for the personnel. The security of the systems was divided into three main classes, from the most lenient to the strictest:

- Class C: Discretionary protection and access controlled systems
- Class B: Mandatory protection with a division into security domains

- Class A: Verified protection

Of these, Class A extended the requirement for verification of security to cover also the system's design and architecture. This classification approach has since been replaced with more flexible schemes, although the United States NIST still advocates for a similar approach in its definition of Minimum Security Requirements for Federal Information and Information Systems NIST (2006).

The Common Criteria epitomizes the normative tradition of software security engineering. Formalizing the security evaluation criteria provides a firm base for evaluation of the *assurance* material: it sets the bar for the material the security process must produce, in order to achieve a particular EAL. The practical usefulness of the Common Criteria, however, remains somewhat dubious: this multi-part publication represents a diffuse tome of arcane security knowledge, unlikely to be utilized unless a level of compliance is specifically required by a regulator. For a software development organization looking to establish or improve their software security engineering function, the Common Criteria should prove useful reference material.

2.1.2 The Common Criteria

The ISO/IEC Common Criteria (ISO Standard 15408) aims to formalize software security, security engineering and security assurance, and to make software security commensurable between applications and organizations. This standard describes the security targets for software products and processes, provides a set of mechanisms to achieve them, and sets the criteria and methodical framework to evaluate the level of achieved security. The standard is published in three parts: the first part describes the general model and the following two specify the functional components and assurance components, respectively. There are two further parts under development: a framework for the specification of evaluation methods and activities, and a pre-defined package of security requirements.

The Common Criteria also is a basis of an international certification organization, the Common Criteria Recognition Arrangement (CCRA), maintaining a three-part public version of the standard (CCRA (2017a)). The CCRA's publication is used by the national certificate authorities to certify a wide range of information technology and software-intensive products, and also as the main source of this section. The public and openly available evaluations, available through their web site, help organizations forming a trusted computing base for their information systems - combined with their own security targets and security objectives. In Finland, the Finnish Communications Regulatory Authority (FICORA) is a member of the CCRA, although as a "consuming member" not authorized to give security evaluations.

Basic Concepts

The Common Criteria describes a thorough formal framework and qualitative criteria for evaluation of a Target Of Evaluation (TOE), an information system or a software entity. The framework is based on a risk evaluation process. The security requirements for the TOE are assessed based on the risk evaluation process. These criteria themselves do not specify, nor restrict, how they should be applied: it is implied that the requirements are mostly normative. In practice, this requires the existence of an ONF, and a derived ANF, as specified in the application security standard.

A central practical contribution of the standard is the concept of Protection Profiles (PP). This concept encapsulates the security objectives and requirements to be completed, regarding both the TOE and its operational environment. The PP also contains the *security rationale* for the security objectives and requirements. The rationale justifies them by offering a set of definitive security assurance: the proof why the security objectives exist, and that a suitable set of security requirements, both for the functional (SFR) and the assurance (SAR) requirements, have been created to meet and trace them.

To clarify the security objectives, a set of Security Targets (STs) is defined. An ST may claim conformance with a number of particular PPs. The ST may also be evaluated to determine sufficient security for the TOE and operating environment. Certain parts of the system are responsible for correctly enforcing the SFR. Together, these form the overall TOE Security Functionality (TSF). The Common Criteria deals explicitly with the functional implications of the security requirements.

The TOE has a number of Security Objectives, which are high-level solutions to identified security problems: these are set by security threats, policies, and assumptions. The Security Objectives are divided into security objectives for the TOE, and security objectives for the operating environment. The requirements for the operating environment affect also the software development process: they may set restrictions to where, how and by whom the software is used, and may either further restrict its functionality or even have additional functionality requirements.

Security Requirements

Security requirements are an essential part of software security engineering. To determine the security requirements, the SFR and SAR are created based on the *security rationale*. SFRs are formally defined and classified: the SFR classification is presented in Table 2.1. The SFRs are required to completely address the Security Objectives of the TOE, at a level of abstraction completely independent from the implementation.

The SAR then describes how the TOE is to be evaluated. In Common Criteria, security assurance exists to provide an exact description of how the evaluation is to be done and to allow comparison between the various STs. The overall ST can be determined for example by combining the security requirements with the claimed conformance with the PPs.

Table 2.1: Security Functional Requirement Classes in the ISO/IEC Standard 15408-1:2009 (2014)

Code	Class Name	Functional Summary
FAU	Security audit	Auditing data, events and analysis
FCO	Communication	Non-repudiation of origin and receipt
FCS	Cryptographic support	Key management and operations
FDP	User data protection	Data and communications protection
FIA	Identification and authentication	User authentication Failure management
FMT	Security Management (normative)	Security roles, attributes and functions
FPR	Privacy	Anonymity and observability
FPT	Protection of the TSF	Technical protection Secure failure and recovery
FRU	Resource utilization	Fault tolerance, priorities, resources
FTA	TOE Access	Access session and history management
FTP	Trusted path/channels	Inter-TSF trusted channels and paths

These 13 SFR classes contain 66 pre-defined security components, specified by CCRA (2017b). The structure, contents and operations of each of the components is specified, with descriptions and notes to guide their implementation. The specification goes on to define the structure of the class formed by its constituent security components. The security components range from specific security patterns ("FTA_MCS: limitation of multiple concurrent sessions") to specific UI functionalities ("FTA_TAH: display to users [..] a history of unsuccessful attempts to access the account").

The software architecture and its operating environment determine which requirement classes are applied. Formally, this happens through the security rationale, justifying and dictating the security objectives, by which the SFRs and SARs are selected. The appropriate functional requirement classes are then derived accordingly.

Security assurance is also extensively covered. The SAR is partly defined by the Evaluation Assurance Level (EAL) of the TOE. The EALs contain an increasing number of requirements ranging from the basic EAL1 to the most rigorous EAL7. The EALs are packages of evaluation assurance requirements with accumulating and increasingly rigorous assurance

requirements. Each EAL has three dimensions: scope, depth and rigour. The assurance classes start from the development process; the components of the evaluation are presented in Figure 2.5.

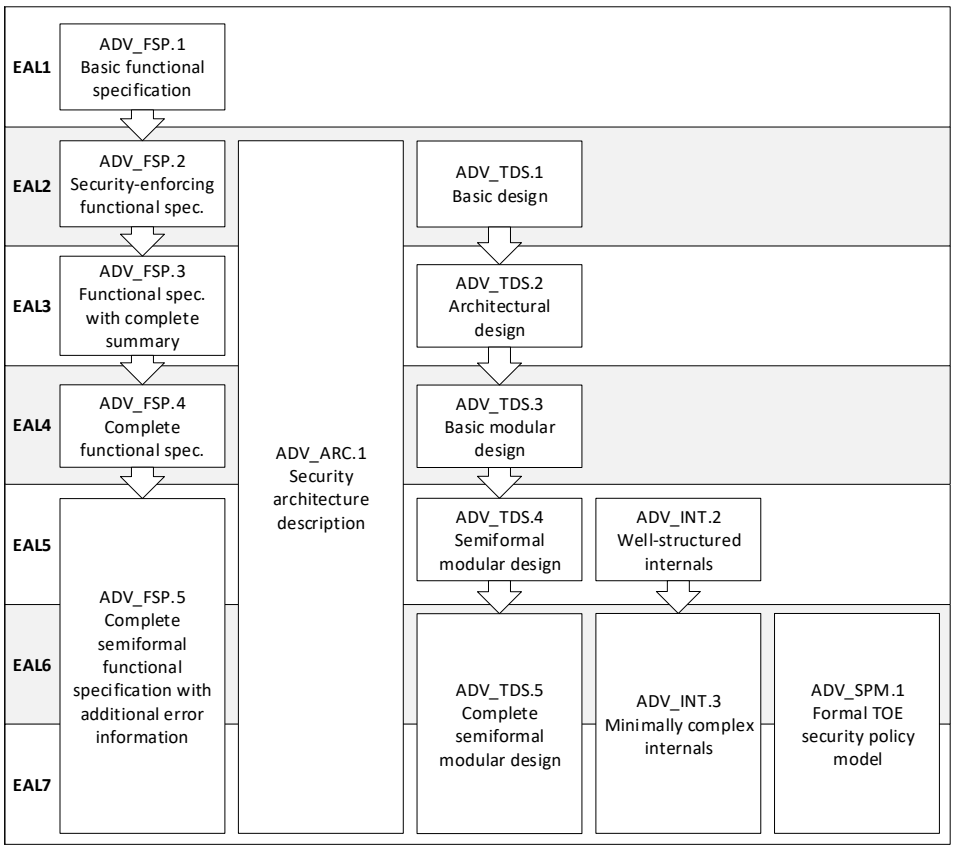


Figure 2.5: The Common Criteria's Evaluation Assurance Levels for Development and the Required Security Assurance artifacts as given in ISO Standard 15408

There are five tracks for Development assurance: 1) Functional Specifications (FSP), with increasing detail and formality requirements; 2) Security Architecture Description (ARC), which is one of the first documents to be produced and is required on EAL2 and up; 3) Technical Design Specification (TDS), also with increasing detail and formality requirements; 4) Requirements for the TOE's internal structure (INT): in software development, this would call for code reviews on the lower levels, and on EAL6 and 7, also code complexity analyses; 5) At EAL6, also formal Security Policy Model (SPM) for the TOE itself is required.

The detailed descriptions and prerequisites for each assurance item are given in the Common Criteria specification Part 3 (CCRA (2017c)). EAL7

does not add new requirements to development assurance, as the added security requirements regard security functionality. In addition to Development assurance, other areas with defined EALs are Guidance Documents, lifecycle support, Security Target evaluation, Tests and Vulnerability assessment. These all have a similar structure of required assurance documentation.

Applying the Common Criteria

The evaluation itself, when applied to a TOE (i.e., not the operational environment part of the Security Target), is of particular interest from the software development point of view. The Common Criteria lists ten specific techniques, most of which are directly applicable, or integral part of, the software development lifecycle. Table 2.2 presents these techniques mapped into the security development life cycle phases.

Table 2.2: Security evaluation techniques in the ISO/IEC Standard 15408-1:2009 (2014)

Phase	Technique
Requirement	Definition of SFR and SAR
Design	Cross-analysis of TOE designs
	Vulnerability analysis and flaw hypothesis
	TOE design analysis against the requirements
Implementation	Analysis and checking of processes and procedures
Verification	Verification of proofs
	Independent functional testing
	Test case and test result review
	Penetration testing
	Verification of processes and procedures
Release	Analysis of guidance documents

These techniques, provided here without detailed descriptions, reflect the Common Criteria's systemic approach to security: after the security work has been done the proof is evaluated. The strongly assurance-based approach is reflected by the distribution of the techniques in the security development life cycle: half of the techniques are applied at the verification phase. Despite an apparent verification phase bias, the design phase techniques are applicable to the implementation phase as well, especially when applied to an iterative development process. It should also be noted that the techniques presented in Table 2.2 are not limited to the evaluation of the software entity (TOE) itself, but are also applied to the evaluation of the operating environment.

In order to efficiently improve the security of a software entity being developed, compatible software engineering techniques are required. These are

to be applied in concert with the security engineering techniques described here. Such methodologies and software security frameworks, specifically aimed for software development, are presented in Section 2.2.

2.1.3 Systems Security Engineering Capability Maturity Model

Capability Maturity Models, such as the ISO/IEC standard 33001:2015 (2015), present an organizational framework for a specific purpose, consisting of processes to achieve it. To measure the process, they also define a set of formal levels of organizational maturity to be reached, assessed using given evaluation criteria. The purpose of a maturity model is to standardize the processes from quality and effectiveness perspective, by making the process predictable and thus risk-reducing. The related Capability Maturity Model (CMM) was developed in the Carnegie Mellon University in 1991 for the United States Department of Defence: the stated purpose of the model was to asses the quality of and capabilities of the defence software contractors.

For software security engineering purposes, ISO/IEC maintains the Systems Security Engineering Capability Maturity Model (SSE-CMM), the ISO Standard 21827 (2018). The standard defines security engineering processes and practices and claims to represent the current best practices in software security engineering. As a semi-formal, prescriptive and risk-driven process, the SSE-CMM lists three security processes:

- Security risk process
- Security engineering process
- Security assurance process

This categorization is represented by the concept of Process Areas (PA), containing the appropriate Base Practices (BP). The *systems* security engineering process in SSE-CMM is formed by 11 process areas, consisting of 61 base practices. These practices claim to cover all major areas of security engineering. The other 11 PAs, addressing the management of projects and organizations, are directly taken from the SE-CMM (1995). The organizational practices are not specific to security engineering, and only the risk management process area is taken into closer examination.

The system security engineering process areas and key work products of the SSE-CMM security processes are presented in Figure 2.6.

The process areas are detailed in the following section: in Section 2.1.3, an overview of the SSE model and the security engineering processes is given; Section 2.1.3 presents the security risk processes, and Section 2.1.3 concerns the security assurance process.

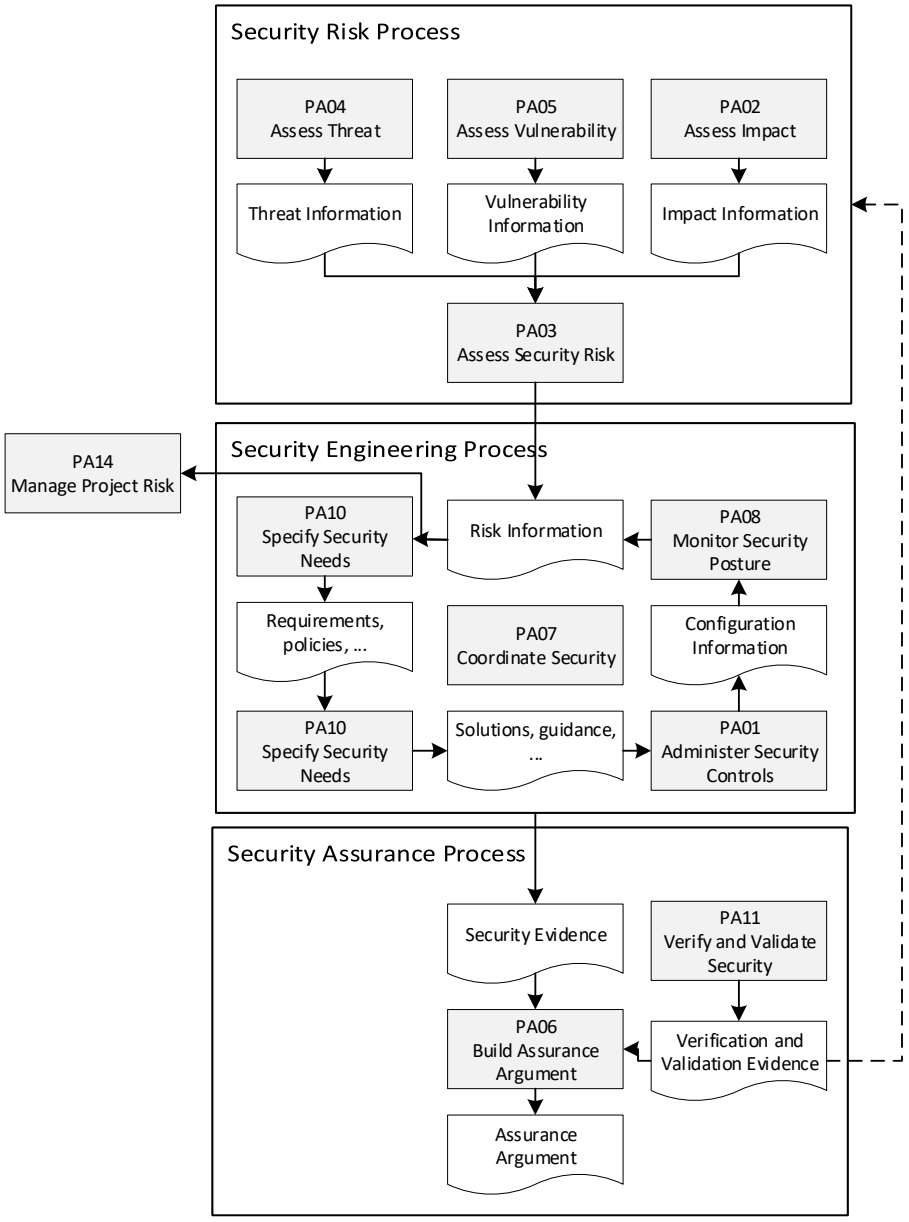


Figure 2.6: SSE-CMM security processes, the system security engineering process areas, and the key work products as presented in ISO Standard 21827 (2018)

In the SSE-CMM, the PAs are structured into five *capability levels* following the Software Engineering Capability Maturity Model (CMM) and other ISO standards such as ISO/IEC standard 33001:2015 (2015) series. Capability levels are defined as follows:

0: Not Performed.

1: Performed Informally.

2: Planned & Tracked.

3: Well Defined.

4: Qualitatively Controlled.

5: Continuously Improving.

The instructions of achieving these levels are generic management practices, having little to do with security engineering as such. The SSE-CMM standard document gives these guidelines in the form of Common Features, specifying the characteristics of the capability levels. The Common Features are implemented by Generic Practices (GP), a checklist of things to get done to achieve the feature.

Based on the descriptions of the levels, the term "capability" itself becomes definable. The first level is producing the necessary work products (Level 1); planning for the production, and tracking the results (Level 2); using an organization-wide and standardized process for planning and tracking (Level 3); establishing quality goals and managing the performance through measurements (Level 4); and, improving the performance by setting performance targets, gathering feedback and piloting new ideas and technologies. Thus, a "capability" is not only doing what you should do be doing in the first place, but doing it by *planned*, *measured*, *standardized*, and *continually improving organization-wide* processes.

The following sections present the contents of the three security processes of SSE-CMM. The division is not clear-cut: there are 11 system security engineering Process Areas, that can be considered to cover the Security Engineering and Security Assurance processes. Although risk assessment is a central concept in information security, in SSE-CMM the Security Risk process is placed among the organizational management practices, although the handling the identified security risks is considered to belong in the domain of the system security engineering. The SSE-CMM risk management base practices are presented in Section 2.1.3. Only one of the 11 system security engineering process areas concerns only assurance; these Base Practices are presented in more detail in Section 2.1.3.

Systems Security Engineering

The SSE-CMM deliberately presents the process areas in alphabetical order "to discourage the notion that the process areas are ordered by lifecycle phase or area". However, for the model to be usable in software development,

even iterative, it has to be applied in a process typically represented by a life cycle. The standard does, however, contain mapping to ISO/IEC Standard 15288 (2015), a standard containing definitions of the software life cycle processes. The generic life cycle phases can be derived through this mapping. Together with the process area descriptions, this information was used to create a mapping of the process areas to a security development life cycle model, presented in Table 2.3.

Table 2.3: SSE-CMM Security Engineering Process Areas

Phase	Process Area
Requirement	PA05 Assess Vulnerability
	PA04 Assess Threat
	PA02 Assess Impact
	PA03 Assess Security Risk
	PA10 Specify Security Needs
Design	PA09 Provide Security Input
Implementation	PA01 Administer Security Controls
	PA07 Coordinate Security
Testing	PA06 Build Assurance Argument
	PA11 Verify and Validate Security
Release	PA08 Monitor Security Posture

Some PAs map to multiple phases in the ISO lifecycle model, but here were placed into a single phase for simplicity. A similar life cycle model is previously used in the software security engineering research, by e.g. Baca and Carlsson (2011) and others, and in the software security life cycle models such as the SDL by (Microsoft (2017)). The simplification is warranted by making it easier to apply into practice, and by enabling direct comparisons between the SSE-CMM and other security engineering models.

In the lifecycle model mapping, the release-time phase is used to cover activities specific to the Operation Environment. Certain activities are considered to precede a new software development project, which would justify inserting a pre-requirement phase into the model. In iterative work, and especially in a continuous deployment organization, the release and pre-requirement phases will eventually merge after the first release. The base practices of the security assurance process area, PA06, are detailed in Section 2.1.3.

The central process area in security implementation is PA07: Coordinate Security (see Figure 2.6 for reference). Under the coordinated effort, Risk Information, produced by the Risk Assessment process (PA03), is used as the primary input. Risk Assessment process in turn gains its input from the Threat, Vulnerability and Impact Assessment processes. Risk Information

is used to Specify Security Needs (PA10), to create the security requirements and policies; these are in turn used to Provide Security Input (PA09), consisting of security solutions, guidance, and other context-dependent documentation. The security implementation is managed by the process area Administer Security Controls (PA01) and includes communication, security awareness and training, and managing the maintenance of the controls. The configuration information, produced by the implementation, and the documentation created during the implementation process, are used to Monitor Security Posture (PA08). This information is then merged back to the Security Risk Information, preferably through re-assessment made according to the Risk process.

The processes specified by SSE-CMM, while abstractly defined, are to be addressed in software development to achieve the specified security objectives. The practices defined in the SSE-CMM cover the requirement specification and analysis phases quite thoroughly. The design and implementation phases rely on the vulnerability assessments and coordinating the security items into the software development processes. Items from security testing (verification and validation), and also from the security assurance and security risk processes, should similarly be used iteratively as an input in software development.

Security Risk Process

In the SSE-CMM, the security risk engineering process acquires its input from assessments made in process areas: assessments of Threats (PA04), Vulnerabilities (PA05), and Impact (PA02). Risk information is then compiled into a risk information document by applying the Security Risk Assessment process (PA03). The created risk information is then fed to the organizational risk management process (PA14), to be used for prioritization, process design and resource allocation by the other organizational processes.

The Threat and Impact assessment processes (PA04 and PA02) are notably similar, both consisting of identification, and defining the metrics and monitoring. Impact assessment is essential in determining the criticality of a system or component: it includes identifying the operational, business, or mission capabilities leveraged by the system, and the system assets responsible for these capabilities and the security objectives. Combined with the impact assessment, this forms the basis of the organizational security risk management process. In threat assessment, the likelihood of identified threats is estimated, and the results are fed to the risk management process.

Vulnerability Assessment (PA05) is distinct to security engineering. In SSE-CMM, a vulnerability "refers to an aspect of a system that can be exploited for purposes other than those originally intended, weaknesses, security holes, or implementation flaws within a system that are likely to be

attacked by a threat". To be adequately performed, the assessment process requires an appropriate level of technical and security expertise, as well as detailed knowledge of the system assets. The vulnerability assessment begins with the selection of the methods, techniques and criteria by which the security vulnerabilities are identified and characterized; the identification of the vulnerabilities is then performed. The assessment itself is then performed, and the identified vulnerabilities added to monitoring.

The resulting vulnerability information is typically a result of paper studies, which in software development should be supported tools, such as a threat modeling tool including vulnerability information (see Microsoft (2017)). In the security risk process, depicted in Figure 2.7, the assessment information is compiled into risk information by the Risk Assessment process (PA03).

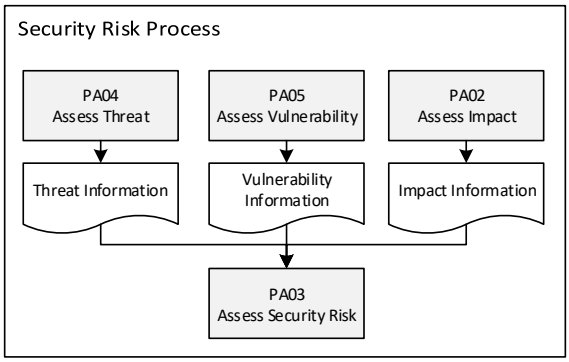


Figure 2.7: SSE-CMM security risk process and the related work products

Outside the SSE-CMM system security engineering practices, the organizational process area PA14 (see Figure 2.6) claims to implement the ISO/IEC risk management standards 15026-1 and 31000. Derived directly from the early 1990s, the risk management process defined in the SSE-CMM is substantially different from the current ISO/IEC Standard 27005:2018 (2018). This newer model is presented and discussed in Section 3.2.2, as applied to software development.

The SSE-CMM specifies the following organizational risk management Base Practices (BP), as adapted from the SE-CMM:

- BP.14.01 Develop a plan for risk management activities that is the basis for identifying, assessing, mitigating, and monitoring risks for the life of the project.
- BP.14.02 Identify project risks by examining project objectives with respect to the alternatives and constraints and identifying what can go wrong.

- BP.14.03 Assess risks and determine the probability of occurrence and consequence of realization.
- BP.14.04 Obtain formal recognition of the project risk assessment.
- BP.14.05 Implement the risk mitigation activities.
- BP.14.06 Monitor risk mitigation activities to ensure that the desired results are being obtained.

The security risk process establishes the risk management model and creates risk information by assessing the known threats, vulnerabilities and their impact and providing means for risk management. Risk information is used by the *security engineering process* to provide means to reduce and contain this risk; in software security engineering, this is typically represented by a software security development lifecycle, examples of which are presented in Section 2.2. The *security assurance process* provides the required degree of confidence that the security needs are satisfied.

Security Assurance Process

The security assurance base practices gather and create the evidence of how the security objectives are met, as well as the potential deficiencies, when compared to the security rationale. The security assurance process is not standalone work: it is linked to the security engineering processes, with the purpose of creating and gathering the required evidence of the conduct and results of these processes. The security assurance process is presented in Figure 2.8 separated from the other security processes.

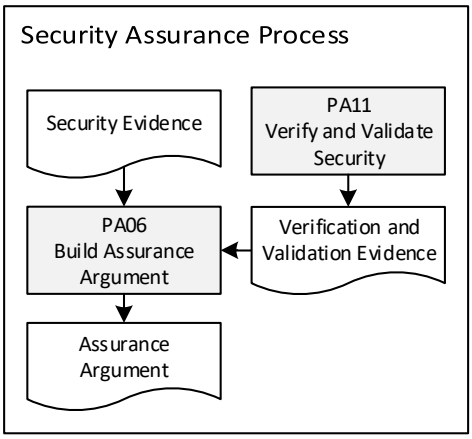


Figure 2.8: SSE-CMM security assurance process and the related work products

In the security assurance process, the process area of Security Verification and Validation (PA11) provides a significant amount of security evidence. In general terms, *verification* answers the question "are we building the system right" and *validation* answers the question "are we building the right system". According to the SSE-CMM, this is achieved by first identifying the solution, e.g., the software being developed; defining the approach and level of rigor for verification and validation; verifying the implementation against the security requirements and needs (i.e.: security objectives); and, communicating the results to other engineering groups.

The SSE-CMM definition of security verification and validation practices is inexcusably vague, especially considering its importance. The standard is only able to provide a generic system testing framework, loosely applied to the context of security. In software development, security assurance engineering consists nearly entirely of various verification and validation techniques; these are discussed in Section 3.2.2.

The security assurance process extends beyond the development-time practice areas. For validation against the security rationale, the central process area is Build the Assurance Argument (PA06). This process uses the input gathered from other processes and process areas: The actual composition of the assurance depends entirely on the ONF and ANF, defined by the security objectives forming the security rationale. The SSE-CMM standard specifies the following six base practices for building the assurance argument:

- BP.06.01 Identify Assurance Objectives: New objectives and modifications to existing ones are identified and communicated.
- BP.06.02 Define Assurance Strategy: Ensure the correctness of security implementation and enforcement.
- BP.06.03 Define Security Measures: Facilitate decision making by collecting, analyzing and reporting relevant performance-related data.
- BP.06.04 Control Assurance Evidence: Maintain the currency and relevancy of the security assurance evidence.
- BP.06.05 Analyze Evidence: Provide confidence that the evidence meets the security objectives and the security needs are satisfied.
- BP.06.06 Provide Assurance Argument: Demonstrate compliance with assurance objectives by presenting evidence, which is then reviewed.

Despite its dependence on the other processes, the security assurance process is equally important as the two others. As part of security management, it provides key stakeholders, such as customers, managers and

auditors, their only viable evidence that the security requirements and objectives have been correctly met.

Security assurance engineering also serves a core technical purpose: for developers, it produces documentation of the solution and its purpose; for operations, it provides maintenance instructions and provisions for the security incidents. Defining the minimum viable assurance, and finding productive uses for the assurance material in the development process, would be an important work management strategy for an agile organization.

Applying the SSE-CMM

Formal conformance with SSE-CMM, even at the basic level, can be an arduous task for a software development organization aspiring to retain the agile principles; this is reviewed in multiple implementations by Silva et al. (2015). The SSE-CMM represents a predominantly deterministic and process-centred development approach. From security engineering perspective, the merits of this approach are clear and the goals universal: a measurable standardized objective-driven process, which effectively produces high-quality results through meticulous, careful and repeatable procedures. The goals are beneficial for security, yet the cost and effects of setting this organization up may be detrimental. This is especially true, when the speed of development is a requirement – in such case as presented in the analysis by Blackburn et al. (1996).

The Common Criteria defines the assurance requirements by giving a set of evaluation criteria; SSE-CMM defines similar criteria for processes necessary to achieve those requirements. When the SSE-CMM is applied in a software development organization, the *value-driven* approach of agile methods is necessarily amended with the *plan-driven* and risk-based approach inherent to the SSE-CMM. These differing approaches are discussed by Boehm and Turner (2003a) and further considered in Section 3.

The process of achieving the higher maturity levels for the individual base practices of the SSE-CMM is a process to be done selectively and in compliance with the security rationale. Even without the pursuit for formal certification of some maturity level, the SSE-CMM and the Common Criteria provide an organization-wide framework for security effort formalization. The level of abstraction leaves the implementation and its evaluation criteria open for interpretation. This allows some freedom of implementation for various organizational contexts (cf. Schweigert et al. (2014)).

2.2 Industry Software Security Models

Building on the tradition formalized in the standards, government and industry organizations have over the decades developed a number of software se-

curity models. The increased demand in software security has led to certain formalizations of these models, varying from documented industry practices in the form of security development life cycle models, to more theoretical and conceptual maturity models.

The currently prevalent models both in research and industry are presented in the following sections. Numerous country and industry-specific models supplement the selection. Together with the standardized models presented above, these models contain the base practices software development organizations perform in the field of software security engineering. The three models selected here are discussed and analyzed for completeness and applicability; they each have their flaws, but when combined to the standardization-derived requirements and methodology definition, should provide feasible in an software development project or organization.

In addition to the three software security processes defined in the ISO standardization, Davis (2005) treats the security management of the organization and projects as an individual fourth process. This separation creates a point of contention for software development: in software development organizations, software *security* development should be an organic part of software development. In practice, the software security frameworks do not contain these management elements. Instead, they focus on the other three processes: security, risk, and assurance. The management practices in contemporary software development are discussed in Section 3.

2.2.1 Microsoft Security Development Lifecycle

Microsoft Security Development Lifecycle (SDL), introduced by Howard and Lipner (2006), offers a set of security practices and tools for the software development process. SDL is a result of Microsoft's systematic program to improve the security of its software products, and the perception that the regular software development methods were not offering adequate mechanisms to create secure software. Development of the SDL followed the considerable publicity of the security issues in Microsoft products, especially the Windows XP, released in 2001. To remedy the security flaws, and in an attempt to regain the lost trust, they established new security policies and a more security-oriented culture, resulting in the SDL. Microsoft advocated the internal use of the SDL by making it a process requirement for several types of applications. The model is to be followed in the development of all software which (a) handles personally sensitive information; (b) is used in academia or enterprises; or, (c) is operated in a networked environment. These categories comprise the majority of current software products. Windows Vista, released in 2007, was the first Windows release to go through the full SDL cycle (Howard and Lipner (2006); see also Anderson (2008)), and Microsoft gives credit to the SDL for the drastic drop of security-related

incidents compared to its direct predecessor, Windows XP.

The activities are distributed into phases in the development according to a traditional life cycle sequence. Later, the SDL was adapted for agile development by Microsoft (2017): in the agile adaptation, activities are presented not only by the phase but also by their frequency during the development. The frequency classes are one-time activities performed once during a project, "bucket" practices performed as seen necessary, and activities performed in every iteration.

The SDL Process

The SDL model, with augmentations for agile development, is presented in Table 2.4.

Table 2.4: Activities in the Microsoft SDL (Microsoft (2017))

Phase	Use in Agile	Activity
Training		Core Security Training
Requirements	One-time	Establish Security Requirements
	Bucket	Create Quality Gates/Bug Bars
	One-time	Security and Privacy Risk Assessments
Design	One-time	Establish Design Requirements
	One-time	Attack Surface Analysis / reduction
	Every-sprint	Use Threat Modeling
Implementation	Every-sprint	Use Approved Tools
	Every-sprint	Deprecate Unsafe Functions
	Every-sprint	Perform Static Analysis
Testing	Bucket	Perform Static Analysis
	Bucket	Fuzz Testing
	Bucket	Attack Surface Review
Release	One-time	Create an Incident Response Plan
	Every-sprint	Final Security Review
	Every-sprint	Certify Release and Archive
Response		Execute Incident Response Plan

SDL activities extend across the development phases, with a focus on the requirement and design activities. Microsoft also provides a rudimentary set of security tools, documentation templates and more detailed instructions for the security work. The accompanying software includes tools for threat modeling, static code review and attack surface analysis, and an analyzer for binaries. The tools and their terminology are limited to Microsoft operating systems, greatly limiting their applicability to general software development work.

Threat modeling is a central practice in the SDL. However, creating and maintaining a sufficiently detailed security model of the TOE and its Operating Environment may provide substantial overhead. To avoid this, a threat modeling tool is to be used in iterative development from the beginning, and the tools is optimally an integral part of the design process. When the concept is turned around, the used design tool should also work as a threat modeling tool.

The main purpose of threat modeling is to make the security risks visible and therefore manageable. In terms of SSE-CMM processes, threat modeling contains the threat and vulnerability process areas. In the original text by Howard and Lipner (2006), SDL threat modeling is a document, or rather, a complex set of documents, consisting of data flow diagrams, a list of protected assets, and list of threats to the system ranked by the risk (implying the use of a quantitative risk management framework). While these documents are considered optional, the most relevant background information consists of use scenarios, a list of external dependencies, security assumptions of the security services offered by other components, and external security notes.

Manually maintaining a threat model in the extensive detail described by Howard and Lipner (2006) is hardly feasible, especially in an iterative process. SDL does, however, give a metric for evaluation of a threat model. An acceptable threat model:

- Conforms to the actual design
- Is dated no more than 12 months ago
- Contains a data flow diagram containing assets, users and trust boundaries
- Details at least one threat for each software asset
- Provides mitigation for all high-level risks

In addition to these criteria, a *good* model adds authentication schemes to the data flow diagrams and provides classification of the threats. An *excellent* model utilizes also a suitable security analysis technique to identify all threats; furthermore, all threats have mitigation, and external security notes include a plan to create customer-facing instructions.

As an example of a security analysis technique to be used in threat modeling, Howard and Lipner (2006) suggest STRIDE to identify six common categories of security threats: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privileges. A number of mitigation techniques are suggested against these, ranging from programming techniques to organizational processes and infrastructure security. For

a software practitioner, techniques usable for threat or vulnerability identification are vital. However, the mitigation techniques may be found beyond software engineering as noted by McGraw (2006).

The list of details and requirements in the original SDL model is staggering. Without clear prioritization, they may not be implementable within the limitations of a smaller development organization, facing acute personnel and time constraints. Fuzz testing is another task difficult to perform in a time-boxed manner. Fuzz tests require expertise, tools, and an extended period of time to produce results of practical benefit.

Applying the SDL

The original model by Howard and Lipner (2006) presents the SDL process as a sequential waterfall-like process, progressing in 12 stages from project inception to product release, and finally, security response execution. The individual techniques are of importance, having an extensive focus on programming techniques and code quality and the validity of the design. Security requirements and the risk process revolve around the software and security design, which itself relies on careful analysis and reduction of attack surface, and creating the design based on threat models. This approach reflects the primary purpose of software security engineering: creating secure software by preventing and finding software bugs during development. At Microsoft, the effort could be labeled "security-driven quality improvement", as security, privacy and reliability were seen, and marketed to the company management under the umbrella of quality (Howard and Lipner (2006) pp. 9-10).

The SDL addresses security training as a prerequisite for all security work. Training is to be complemented and improved as necessary during the development whenever changes in requirements or design warrant for it. A noteworthy requirement-phase practice is setting the quality gates called "bug bars". Essentially, this is a quality assurance technique involving the setting of a bug standard, and the triage for placing the found bugs into three categories based on the risk assessment. The process also involves setting clear error categories for bugs that would necessitate a fix or further investigation, especially when a fix is *probably* needed. These categories are separately set for both client and server side software and are largely results of fuzz testing (see Howard and Lipner (2006) pp. 163-164). In the SDL, security testing techniques also have a focus on the improvement of code quality, in addition to producing test reports for the security assurance process.

SDL's adoption for agile development is a late addition to the model. In the original form by Howard and Lipner (2006), the SDL is described as a process, with 12 distinct stages. The instructions for integrating the process

with agile consists mainly of recapitulating the agile principles and Extreme Programming quality improvement techniques. In the current model by Microsoft (2017), the main changes are simplifying the process into six phases, and adding superficial labeling to the activities according to their perceived frequency during an iterative project.

If followed literally, restricting some activities' use frequency to a strict one-time limit may even be detrimental for security. In iterative work, no activity can be guaranteed to be limited to simple one-time execution. This thinking appears to originate from the model's "waterfall-like" origins: for example, the model instructs risk assessment to be done only once during a project. In iterative development, it is essential for the risk assessment process to be performed after each change in relevant design or requirements. Finally, Microsoft's original online documentation for the agile model created confusion by treating an agile iteration as a synonym for release. This echoes the early agile ideal of each agile iteration producing a potentially shippable product release.

Much like the SSE-CMM, the abstraction level of the instructions in SDL is high. The model captures the essentials of software security engineering and assurance processes and promotes the use of threat modeling as a way to manage security risks. The SDL model does not, however, provide a standalone security solution for security implementation, and in practice provides a loose framework of practice categories. In addition, the central concept of a threat model will require either a very competent tool, or a very dedicated security-minded person to get created and maintained at an acceptable level.

Howard and Lipner (2006) do describe the rationale, and much of the practical reasoning and even the process of building up the SDL activities, as it was done at Microsoft. It is conceivable, however, that many of the conditions and motives are hardly applicable to more modest-sized organizations, with perhaps more meagre resources. Applying SDL, more detailed models should be consulted, including the Common Criteria for evaluation and the SSE-CMM for implementation. Security rationale justifies the security means.

2.2.2 Building Security In Touchpoints

The Touchpoints by McGraw (2006) also utilizes the lifecycle approach to software security, similar to the SDL. It essentially consists of the three security process areas as in the SSE-CMM. The root of this methodology are the seven key techniques for implementation of secure software, or the *Touchpoints*, addressing seven identified problem domains susceptible to exploitation in software development.

These techniques were first introduced by Howard (2004), the same

source that presented the Microsoft SDL. Touchpoints conform with other security and software development lifecycle models and can be seen as an independent extension and enhancement to the SDL, with many practical insights how to implement it. The activities included in the Touchpoints are presented in Table 2.5, divided by the development lifecycle phase. For mutual comparability, artifacts affected by each activity are derived from the Common Criteria: these may be SFR documentation, SAR documentation, the TOE itself, or the Operating Environment (see Figure 2.5).

Table 2.5: Touchpoints

Phase	Touchpoint	Target
Requirements	Security requirements	SAR and SFR
Design	Architectural risk analysis	Security arch. description
	Abuse cases	SFR
Implementation	Code review	Source code
Verification	Risk-based security tests	TOE
	Penetration testing	TOE, Operating Env.
Release	Security operations	TOE, Operating Env.

Code reviews, the sole activity to be performed in the implementation phase, is identified as the most important and effective security practice of the seven Touchpoints. The importance of code reviews is supported by e.g. Glass (2003), although Glass stresses that there is no single method to remove coding errors: a combination of methods, tools and techniques is required. Implementation errors are the traditional center of attention in software security, categorized into a taxonomy by Tsipenyuk et al. (2005). Software quality issues introducing exploitable security vulnerabilities, as well as the difficulty of tracing security bugs through traditional quality assurance methods, have been a main cause for creating the current security development models and methodologies.

The Touchpoints places the rest of the practices into order by their effectiveness, from most to least effective. This order is Architectural risk analysis; Penetration testing; Risk-based security tests; Abuse cases; Security requirements; Security operations. When examined in the context provided by the Common Criteria, as presented in Table 2.5, the practices affecting the TOE directly are ranked more important, and practices targeting the Operating Environment are given less importance. This is consistent with the idea of building security into the product, not "bolting it on".

The implementation-specific part, as written in 2006, held the following areas that require special attention: 1) Input validation and representation; 2) API abuse; 3) Security features; 4) Time and state; 5) Error handling; 6) Code quality, and; 7) Encapsulation. These conform with the security engi-

neering base practices in the SSE-CMM presented in Section 2.1.3, although differently named and formalized as processes.

To effectively address the security threats, security engineering requires a risk definition and management process. Without domain-specific knowledge the risk analysis cannot produce tangible and usable results. The book defines five stages for risk management but advises referring to external sources for more refined risk management processes. The five-phase model presented by McGraw is presented in Table 2.6. This model is a rudimentary version of ISO/IEC Standard 27005:2018 (2018). It does, however, capture the key elements of risk management.

Table 2.6: BSIMM Risk Management Framework by Synopsys Software Integrity Group (2018)

Level	Activity	Actions
1	Analysis	Understand the business context
2	Identification	Identify the business and technical risk
3	Prioritization	Synthesize and rank the risks
4	Mitigation	Define a risk mitigation strategy
5	Implementation and verification	Carry out fixes and validate

The risk management framework is based on an *analysis process*, requiring knowledge of security issues combined with knowledge of the business domain. After composing a list of identified security risks, the risks are prioritized. The priority scale can be as simple as High, Medium or Low: use of a context dependent risk prioritization framework may be necessary. The *risk mitigation strategy* is the result of the prioritized risk list, which is analyzed and mitigation methods selected based on the business and technical risks, the TOE and the Operating Environment; the product of this analysis is a Risk Analysis Report. In the implementation and validation phases, the risk mitigation methods are verified and the fixes are carried out as necessary; here McGraw's framework is very testing-oriented, and the verification is generally compared to following a test plan. The framework goes on stressing the importance of metrics and measurability, calling for quantitative decision support. This can be achieved by setting up formal criteria for measurement by case, or using software-based tools with predefined measurement criteria.

The methodological framework presented by McGraw, which includes Touchpoints, is a collection of best practices and tools. It was one of the first software security engineering methodologies and continues to be relevant to both industry and research. McGraw also has helped creating the *Building Security In Maturity Model* (BSIMM) by the Synopsys Software Integrity

Group (2018). The BSIMM model is updated by annual industry surveys about security activities and provides central statistics about the industry's software security initiatives. The primary contribution of the BSIMM is highlighting the current issues in software security, and actively reporting the industry's best practices on software security.

2.2.3 OWASP Security Assurance Maturity Model (SAMM)

The Open Web Application Security Project (OWASP) consists of several sub-projects ranging from implementation and testing frameworks to security verification and metrics. It also contains a large number of open-source guides and frameworks. Most relevant of these from the software engineering point of view is the Software Assurance Maturity Model (OWASP SAMM (2017)). As the name suggests, SAMM has many aspects outside the direct scope of software security engineering. It does, however, include an SSDLC model, which it largely inherits from OWASP's now discontinued Common Lightweight Application Security Project (CLASP). The structure of the SAMM is presented in Figure 2.9.

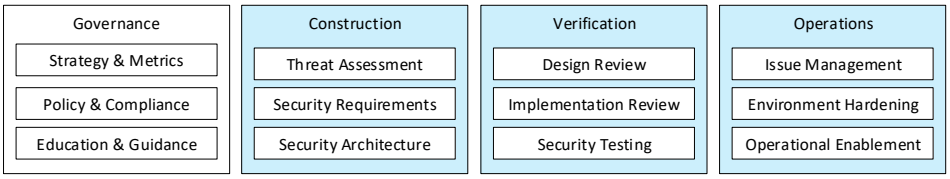


Figure 2.9: SAMM Business Functions and related Security Practices

The structure of the SAMM is highly symmetrical: in SAMM, the four business functions are each divided into three security practices, each with three security objectives. Each of the 36 ($4 \times 3 \times 3$) security objectives is achieved by executing two main security activities, resulting in a framework of 72 activities. In the assessment scheme, these activities are scored based on the scope and frequency of their implementation, resulting in one of three measurable maturity levels. As the name of the model suggests, the majority of the practices and activities concern creating and verifying security assurance. Overall, the SAMM primarily concerns the management of software security engineering, with many practices extending beyond the development activities.

The overall abstraction level in SAMM is quite high, and the style is descriptive. SAMM leaves the details of implementation open and reliant on other, more technical security instructions. The selected approach appears feasible and applicable to a wider variety of software development organizations, although the software security engineering techniques themselves are typically not explicitly specified. A precondition for utilizing the

SAMM model is the existence of security risk and requirement management frameworks. SAMM, however, does not specify the outlines for these, or instructions for their construction.

Governance

Governance in SAMM is managing overall software development activities. The three subcategories of the Governance function are the most important in setting the security objectives: Strategy and Metrics, Policy and Compliance, and Education and Guidance. The first two are also the primary practices used in setting the security assurance target. In the ISO standardization, these would represent the application normative framework derived from the organizational normative framework, and the target assurance level results from these. The governance objectives concern strategic planning of security effort by understanding the assets to be protected and their business value, managing the security processes and expenditures based on the risk and value, and establishing the compliance requirements from the normative context.

Education takes the form of security training and personnel certification schemes. Education includes not only basic security awareness, but also establishing technical guidelines, introducing the security specific roles and role-based training, security coaching of the software teams, and creating a role-based examination and certification program. The education scheme is especially essential to personnel responsible for security architecture.

The creators of SAMM have for some reason chosen to explicitly mention the possibility of penalizing staff for making bad business decisions regarding application security. The penalty is not specified, but enforcing security by punishment schemes is shown to have an *adverse* effect on security behavior by Herath and Rao (2009).

Construction

SAMM's definition of construction is defining goals and creating software within development projects. The Construction function is most relevant to technical security engineering. It consists of Threat Assessment (TA), Security Requirements (SR) and Security Architecture (SA). TA in SAMM refers to project-level risks, which are "accurately identified and characterized potential attacks upon an organization's software in order to better understand the risks and facilitate risk management" as specified in OWASP SAMM (2017). While this conforms fully with the SSE-CMM process model, no instructions are given on how the threat modeling should be conducted, or how the risk is to be assessed. The assessment criteria for Threat Assessment is based on the existence of the used practices and the frequency of

their use; the quality of the assessment is not considered.

According to SAMM, Security Requirements need special attention and promotion to get implemented. The objectives for SR aim to explicitly consider security during the software requirement process, derive the security requirements from risk analysis and the business requirements, and to spread these practices to all software projects and third-party dependencies. Assessment of the SR practices is based on team-level activities, and their specific usage of best practices and compliance guidance, and the use of access control matrices where relevant – access control being the only security engineering technique explicitly mentioned. SR audits for design and requirement documentation are recommended for each *development iteration* before a release is made. SAMM commendably promotes the review of vendor agreements as an SR-related security activity, suggesting similar security review and testing policies to be extended into the third party SW development as well.

Security Architecture objectives in SAMM appear to conform with many of the generic software architecture principles and objectives. The architectural guidance for software design is quite generic, such as applying security principles to design or identifying security design patterns from the architecture. For the more explicit architecture objectives, SAMM suggests the following activities:

- Maintain a list of recommended software frameworks.
- Identify and promote security services and infrastructure.
- Establish formal reference architectures and platforms.
- Validate the usage of frameworks, patterns, and platforms.

Implementing these activities requires considerable skill in both software engineering, i.e., programming, and component-level security. For example, maintaining a list of recommended software frameworks—a first-level activity in SAMM—is not a simple feat to accomplish. Identifying a list of recommended software frameworks would be most useful for the security services and infrastructure, identified by executing the 2nd-level security activity. The ambitious 3rd-level activities resulting in formal and validated reference architectures, patterns, and platforms rely on this framework list as well. In essence, achieving the first maturity level involves majority of the work, with the benefits manifesting themselves only at higher maturity levels.

Software frameworks can be dynamically updated, so extending this activity to the code level is not feasible. SAMM promotes standardization and fixed component-level design guidance, based on careful inspection of each

component's incident history and the track record in its response to vulnerabilities. Non-security considerations are e.g. assessing the functionality and the level of complexity of the implementation. From the model, it is not directly evident at what maturity level these checks should be done.

Verification

Verification in SAMM is defined as checking and testing artifacts produced throughout software development. Verification subprocesses are divided into three categories: design review, implementation review and security testing. Artifacts are defined as design documentation, software configurations, and the source code.

The concrete design requirements missing from the architectural objectives are placed under the design review objectives instead. The aim is to allow and enable the project teams to perform self-checks and utilize lightweight reviews for the design. In SAMM's three-level maturity scale, the security team is conducting design reviews only at the strictest level. Concrete security engineering activities consist of identifying the attack surface, analyzing the design against security requirements, and developing data-flow diagrams for sensitive resources. Data flow diagrams can be considered part of either software design or a threat model, leading in an apparent inconsistency between SAMM requirements: while the form of threat modeling was not clearly specified, requiring data flow diagrams at design review appears peculiar.

On the security management side, the suggested verification activities are organization-heavy, and call for deploying a design review service for the software projects and establishing release gates for design review. Both of these practices can be seen to lead to organizational bottlenecks in software production, causing development and deployment delays. Agile development and management may still be effective in micro-managing these changes to mitigate the adverse effects on productivity (Karlstrom and Runeson (2005)).

Implementation reviews include code reviews and analyses, based on review checklists, recognition of security critical code and use of automated tools. SAMM does not reflect the current state of the art in the sense that the basis of implementation review is seen to be manual, augmented with tools. In an agile organization, continuous integration and unit testing is the key of keeping the code base clean of bugs and unspecified functionality; integrating an automated source code review tool would be a logical enhancement to the CI and testing tools. As a quality control issue, SAMM suggests the use of release gates for implementation review. It also suggests applying a straightforward passing criteria for the reviews, with only one or two vulnerability types. However, in order for the review not to just

be a formality, the criteria should actually represent the security risks the application is facing. From the technical point of view, it would be more advisable to deviate from SAMM a bit, and (a) use automated tools from the start and (b) utilize the tools to make the review as thorough as possible. Several reviews may also be performed with focus on specific vulnerability types if the development resources allow for it.

The third verification practice in SAMM is security testing. Of security testing methodologies, only penetration testing is specifically mentioned. Penetration testing is instructed to be used before each release. The test cases themselves are to be derived from the security requirements. The use of automated tools is also promoted in security testing, a practice naturally suitable for agile software development. SAMM does, however, also suggest another security-related release gate for security testing. Again, automation and agile practices could mitigate the productivity reduction resulting from these measures.

Operations

In SAMM, operations stands for the management of the created software releases. SAMM specification conforms to the traditional release-based thinking, as opposed to agile continuous delivery models such as DevOps. This approach is also visible in lack of utilization of automated security tools. SAMM's operations security practices are issue management, environment hardening and operational enablement.

Issue management consists of management practices, such as establishing security response teams and issue escalation points, and creating response processes and incident disclosure processes. Technical practices, consisting of root cause analyses and collection of per-incident metrics are built on the incident management; these practices are very closely linked, and often managed by interconnected tools.

Environment hardening practices are routine procedures to conduct in any operational environment maintenance: documentation of the used components, security patching, and monitoring the environment configuration status. Hardening practices, according to SAMM, also include deploying relevant operations protection tools, such as firewalls, anti-tampering measures, integrity verification tools, and secure logging solutions for security forensics. The operating environment should also be audited, targeting up-to-date environment specifications, monitoring tools, configuration management, and the use of protection tools listed above.

Of the operations practices in SAMM, operational enablement appears to be closest to continuous development models. The objectives are in the improvement of communication between development and operations teams, and improvement of the operational maintenance practices. The communi-

cation is improved by increasing the development-time documentation and making it available for the operations team. The additional documents include identification of software errors and alerts, and recovery procedures for critical error situations. The operations team should maintain an operational security guide, done in cooperation with software development teams to avoid deviation of security practices. For further security assurance on SAMM level 3, the environment should be audited before each release, adding yet another security gate into the deployment process. A final gate is the code signing process, performed only for the relevant parts of the software. This process is also to be performed during each release.

2.3 Summary

The security engineering methods and models presented in the above sections represent the current state of the art in generic software security methodologies and standards, also forming the core topics of software security engineering research. The selected models and standards are presented in Table 2.7.

Table 2.7: Software security engineering models and standards included.

Type	Name	Source
Standard	Application Security	ISO 27034
Standard	The Common Criteria	ISO 15408
Standard	SSE-CMM	ISO 15443
SSDLC	SDL	Microsoft
SSDLC	Touchpoints	Synopsis/Cigital
Maturity Model	SAMM	OWASP Foundation

The table presents the standards, software security development life cycles and maturity models currently most relevant to generic software development. A systemic literature review of the methodologies used in software security engineering is pending; one of the most comprehensive method studies was performed by Win et al. (2009). In many respects, this study is in need of an update, indicating a research gap.

Several branches of business and technology have their own security regulation supplementing the generic practices presented here. The most prominent examples are the health care, and financial sectors, and the defence industry along with many other governmental, societal, and economical areas. This regulation, however, has its roots in the ISO software security standard framework presented here. The same principles are also used in the creation of the mechanisms to protect personal privacy in electronic systems.

The main purpose of security standards is to gain commensurable security and security assurance by setting objective evaluation criteria and providing explicit security processes and assurance requirements. In practice, none of the standards alone is capable of providing this. Only when thoroughly examined and combined, the three presented ISO standards provide the framework and the criteria for just that: Application security provides the framework for security rationale, objectives and requirements, and the security management. SSE-CMM describes the processes to build the required security and assurance; the Common Criteria sets the evaluation criteria for the product through the Protection Profiles.

The examined industry models provide lightweight frameworks for managing and measuring the security work. However, the main framework, against which the work is to be analyzed, is set by the security standards. Section 3 sets the security frameworks into the context of agile software development. This section also provides more insight into the current state of the art in both industry and research. For each of the examined areas, gaps in both the practical implementation and scientific knowledge are pointed out.

Chapter 3

Agile Security Development – Methods and Solutions

In this section, the key methodologies used in agile software development are presented, and their applicability to security engineering is analyzed. The knowledge base and the analytic framework is drawn from the field of software security engineering presented in the previous section. The selected agile methodologies and their core techniques are presented in Section 3.1. Based on the analysis, an agile software security engineering framework is presented in Section 3.2.

Combining the SSDLC and maturity models with agile software development requires combining the business and security objectives to set the objectives for development. This requires a rigorous approach to risk management, and is the topic of various system engineering theories, ultimately being rooted in mathematics on the technical side. Elicitation of the security requirements, including the management of security risks, has to be conducted in a way that contributes towards achieving the security objective without unnecessarily causing hindrance to the business objectives. This section presents ways to 'enhance' the software development process with software security engineering activities.

3.1 Agile methods

The paradigm change that began in the 1980s and 1990s was summarized and even aggravated in the Agile Manifesto by Beck et al. (2001). Since the manifesto, the agile methods and principles have received a near-universal adaptation in software engineering, as verified by studies by Rodríguez et al. (2012) and Holvitie et al. (2017), and industry surveys by VersionOne (2018). The development of methodologies, leading to agile development models, has its roots in the age before computers. Outside the field of software engineer-

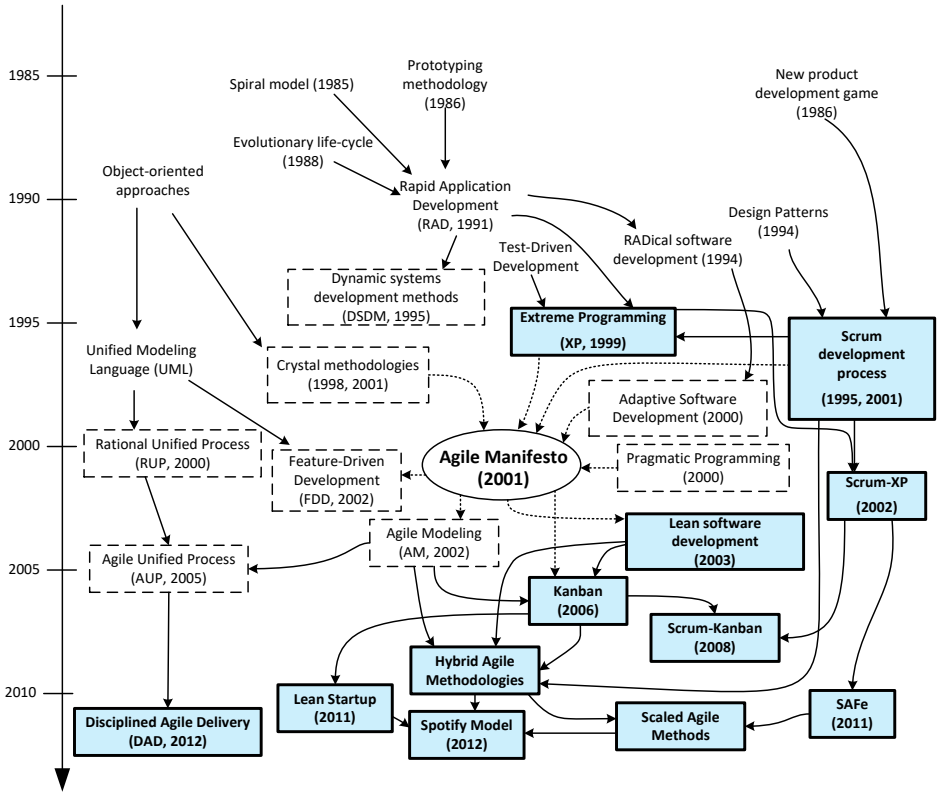


Figure 3.1: Evolution of currently prevalent agile methods improved on Abrahamsson et al. (2003)

ing the lightweight, iterative and incremental management and development models have a much longer tradition and many predecessors, such as the use of Kanban with the Toyota Production System, as described by Sugimori et al. (1977). Iterative models have been prevalent especially in the methodologies aiming to improve quality control.

The evolution of these methods was recounted and illustrated by Abrahamsson et al. (2003). Building upon that work, Figure 3.1 presents the development from the 1980s onward in a cleaned and modified form, updated to include the recent developments in the methodologies and their origins.

The diagram, while necessarily simplified, contains the historically significant techniques and methodologies that are either still in use, or have had a marked influence on the currently used agile methods. The diagram is expanded to include the developments in lean and hybrid methods, scaled agile methodologies and the inclusion of new hybrid methodologies. Also, the

development branch originating from the Rational Unified Process (RUP) published by Kruchten (2000) leading to Disciplined Agile Delivery (DAD) by Ambler and Lines (2012a) is included; the latter is an exercise in enforcing pre-planned activities into iterative and incremental development processes – as such, a prime candidate for security projects, although notably breaking away from the agile principles and values. The Test-Driven Development model, a direct precursor to the XP method by Beck (2000), was also absent in the original source and was thus added.

In Figure 3.1, the shaded boxes represent the agile methods with reported use of at least 1% in the survey by VersionOne (2018). The trend indicates increase use of hybrid methods, developed to provide scalability and enable synchronization of the work of parallel development teams. The non-boxed elements in the figure represent past methodologies or techniques preceding the "true" agile methods. The Agile Manifesto, marked with an ellipse, is included to provide a timeline for the methodologies and their influences. The elements that have a dashed border, represent methodologies or tools that have acted as an influence, but no longer have significant reported use.

The vast majority of current software engineering organizations is using software development methods that can be labeled agile. Adaption of agile development processes, methodologies and models into software security engineering has been researched relatively poorly. Most of the work concentrates on combining specific agile methodologies with e.g. requirements of the Common Criteria, or the SSDLC methods, both described in Section 2.

The following sections present the basics of the most used agile methodologies and the key concepts in agile methods. The agile methodologies selected for review are Scrum, Extreme Programming, Kanban, and the scaled methods. Key agile concepts are presented as provided in the Scrum method in the next section. These concepts include iterations (sprints), the main artifacts (backlogs), and the *definition of done*. This important concept, the definition of done, marks the completion of a process starting from requirements gathering, continuing with design and implementation, and finally, fulfilling the verification criteria and achieving acceptance.

3.1.1 Scrum

Scrum is one of the earliest and currently still the most used agile methodology. Scrum was defined in its current form by Schwaber (1995), and revised by Schwaber and Beedle (2002). The term 'scrum' itself originates from "The new product development game" by Takeuchi and Nonaka (1986), as does much of the game-themed terminology used in earliest agile methodologies. The Scrum processes are divided into two iterative stages, presenting the "pre-game" planning and architecture stages on the left, and compressing the "game" and "post-game" stages onto the right, consisting of production

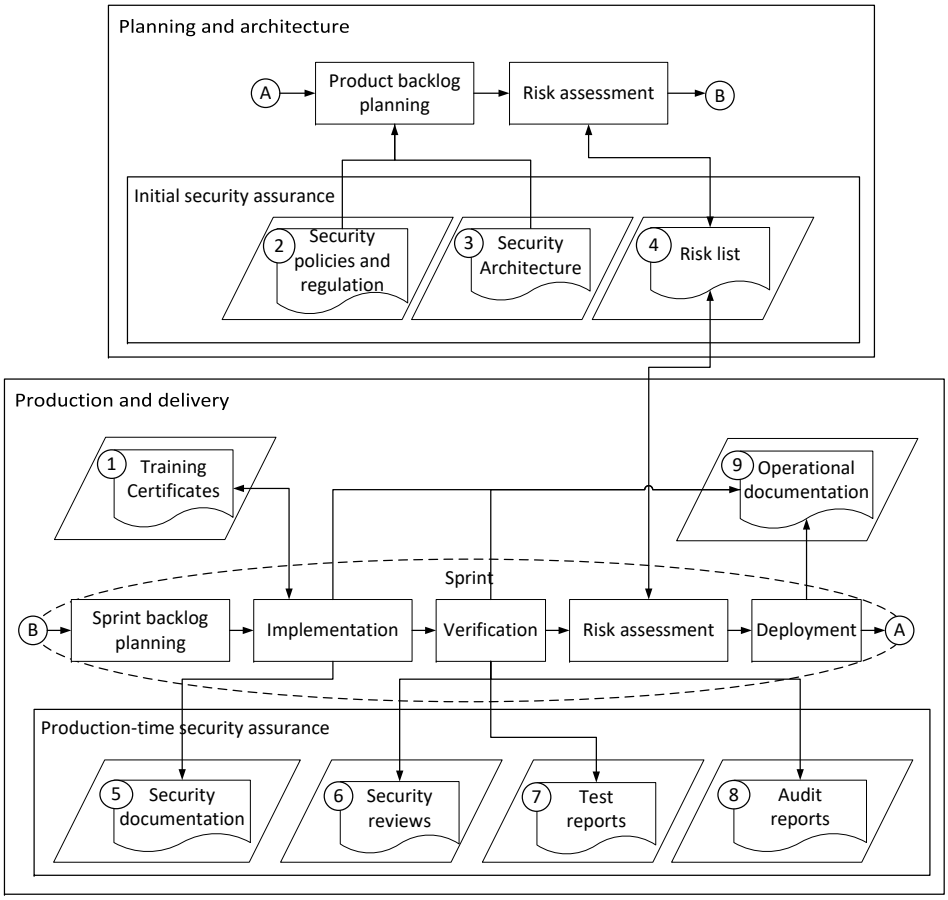


Figure 3.2: Software security engineering processes executed in Scrum framework by Schwaber and Beedle (2002): roles, events, and documentation artifacts

and delivery. The iterative Scrum process, as related to various security assurance artifacts, is presented in Figure 3.2.

Numbered items refer to the security assurance artifacts created or utilized during a Scrum iteration, linked to the processes and events. Of these, (1) training and (2) security policies typically exist prior to the start of the development process; organizations may also have a security risk assessment baseline to perform initial risk analysis. Training requirements may also emerge and be addressed during implementation.

Security architecture (3) is one of the most common security assurance requirements, essentially a part of the general software architecture with security requirements included. Item (4) is the security risk management documentation: risks are related to the protected business value, and the risk mitigation techniques are to be translated into concrete backlog items.

The prioritization of these items can also be based on the risk assessment. The rationale for the security work is based on the security objectives, which are derived from the initial security assurance documentation.

Items from (5) to (9) are produced or updated during the development process through documentation (5), code and software reviews (6), and various forms of security testing (7). In some cases, also security audits, performed by a certification body or other third party, may be required and an audit report is produced (8). Before release to production, instructions for maintenance and operations are produced (9). These are sometimes supplemented by a plan, or roadmap, for updates. This, in turn, is based on the architecture and the technical documentation. To provide some context, in the Common Criteria item (5) would represent the assurance documentation stack ADV_FSP.x (see Figure 2.5).

Scrum introduces a flexible framework, rather than a strict methodology, within which the techniques and processes most suitable for the task at hand can be employed. In Scrum, work is completed in short iterations, called sprints. In its basic form, Scrum introduces a simple framework of three roles (scrum master, developer and the product owner), three core artifacts (product backlog, sprint backlog, product increment), and four events (sprint planning, daily meetings, sprint review, and sprint retrospective). When scaling up the team size or the number of parallel teams, a sprint burndown chart is a practical necessity to track and coordinate progress of the work.

The Scrum workflow captures the essence of agile software development work. In Figure 3.2 security specific work stages and security assurance have been added to a rudimentary Scrum process. The security engineering processes, presented in Section 2, practically introduce security and quality gates into all the stages: product backlog planning involves architecture reviews; risk assessment itself resembles a quality gate. Sprint backlog planning implies design and test case reviews; in the implementation phase, there are reviews for the code, tools, and programming practices and processes; the verification stage introduces further security testing and possible third-party audits. The release, done after a final risk assessment, introduces security documentation creation and consequent reviews, and the possible certification of the code.

Requirements are elicited by customer collaboration (i.e., the product owner), and are communicated in the form of stories. The main difference between a story and a requirement is the level of formality and detail: the user expresses what feature is wanted or which objective needs to be achieved, and these are formalized into requirements and further into development task in backlog planning events. Items in the product backlog are prioritized together with the product owner, and the actual commitment to implementation is done by the developers in sprint backlog planning. Com-

mon ownership of the items is promoted, meaning that the team commits to implement each task they select for the backlog of the upcoming sprint. Each backlog item is assigned a formal definition of done criterion, verified through a continuous (automated) testing effort.

In practice, Scrum is a set of rules tying the above elements together and governing their use throughout the project, adapting to the changes occurring during the process. The incessant drive for 'value' is inherent to not only to Scrum, but nearly all modern software engineering. This approach has the potential downside of causing functionally complete software to be released for production use despite inadequate, or completely missing security engineering and validation.

3.1.2 Extreme Programming

Predating Scrum, the *eXtreme Programming* (XP) by Beck (2000), has been influential in the definition of later agile methodologies and, most importantly, the agile manifesto itself. XP has a more holistic view of the software lifecycle, not concentrating only on the development phase. In XP, the development lifecycle is divided into five phases, from exploration through planning, iterations to release, 'productionizing' and maintenance to 'death'. Despite the unconventional naming, the phases constitute a common lifecycle model with XP activities defined for each.

Extreme Programming aims at increased code quality while improving productivity, and suggests achieving those goals through four activities: coding, testing, listening and designing. *Coding* is the act of implementation and production of the software; the software will be subjected to rigorous *testing* both on unit and integration levels as part of quality assurance. Current practices of continuous integration and continuous delivery (CI/CD) reflect this thinking.

In Extreme Programming, *listening* is a generic term for requirement elicitation done in cooperation with the software's customer, and implemented by a process called *the planning game*. The planning game is a mechanism similar to Scrum's backlog planning and is divided into *release planning* and *iteration planning*. This mechanism of requirement elicitation translates the user requirements, recorded in story form, first into concrete requirements in cooperation with the customer, and then into implementable and testable functionality (tasks) within the development team. The planning game process is divided into three phases: exploration, commitment and steering. In release planning, exploration deals with stories, which get estimated and, if necessary, split into smaller entities; commitment phase involves sorting of the stories. Sorting is based on value and risk: in this context, the risk is not taken to mean a security risk, but rather level of uncertainty dealing with the completeness of the story: this means the likelihood of the story to

change (volatility) and the complexity of the story's implementation.

From the viewpoint of software security, security-related stories create both explicit and implicit business value, which the team should have the ability and expertise to estimate. After estimation, the team assesses the *velocity* with which they can perform, and choose the *scope* of the items that will be included in the next release. In iteration planning, the items in the scope of release are translated into tasks and assigned to individual developers using a similarly staged process. An industry survey by Licorish et al. (2016) indicates that the planning game is one of the least utilized agile activities with 27.7% adoption rate.

XP also introduces other important techniques, such as pair programming. This is to be applied using value-based guidelines, such as sustainable development pace enforced by a 40-hour upper limit to the work week. Additionally, the implementation effort should be test-driven, emphasizing the testability of all planning items. However, for security requirements, expressing requirements as test acceptance criteria may not be a suitable approach. XP also discards many of the planning phase practices, promoting an emerging design and architecture to reduce the re-design work.

The concept of emerging design is also a source of criticism targeted to all agile methods – not only XP (see e.g. Bellomo et al. (2014)). The criticism is on the other hand supported by reported 74.5% adoption rate of "Simple Design" as surveyed by Licorish et al. (2016). This approach, combined with the low adaptation of planning activities, provides ground for hypotheses against the use of agile methods in security critical work. In contradiction to the anti-agile hypotheses, in a case study by Adelyar and Norta (2016) the use of XP and other agile methods was found directly beneficial in the mitigation of security vulnerabilities. Early software security engineering focused on XP due to its prevalence in software development in the early 2000s. Security engineering in the XP method has been a central topic in research by e.g. Wäyrynen et al. (2004) and Karlstrom and Runeson (2005); requirement engineering in XP has been researched by Boström et al. (2006). Security assurance in XP was discussed by Beznosov and Kruchten (2004), and a study about security processes in XP was conducted by Ge et al. (2007). A common theme in the findings in these research reports is the practical adaptability of agile methods – in this case, XP – to virtually all types of security engineering processes. This can be generalized to other agile methodologies as well, indicating that the current research gap is in the security techniques and tools, and the management of security engineering work in the agile projects.

3.1.3 Lean methods and Kanban

The *Kanban* method combines agile principles with a lean approach. The lean approach, introduced to software engineering by Poppendieck and Poppendieck (2003) among first adopters, has its roots in the manufacturing industry. Lean thinking has been widely adapted to software engineering (see e.g. a systematic literature review by Ahmad et al. (2013)), and Kanban is among the most used management methodologies (VersionOne (2018)). In software engineering, lean methods promote high-quality work, knowledge creation, and deferring commitment and decisions until they are absolutely necessary.

The Lean Software Development deals extensively with the concept of 'waste'. In the broad definition, anything that does not produce value for the customer is considered waste. Lean development extends the agile framework by adding seven key principles. These principles, with their support tools, are given in Table 3.1.

Table 3.1: Lean Software Development principles and support tools as given by Poth et al. (2018)

Key Principle	Support Tools
1. Eliminate Waste	Seeing waste
	Value stream mapping
2. Amplify learning	Feedback
	Iterations
	Synchronization
	Set-based development
3. Decide as late as possible	Options thinking
	The last responsible moment
	Making decisions
4. Deliver as fast as possible	Pull systems
	Queuing theory
	Cost of delay
5. Empower the team	Self-determination
	Motivation
	Leadership
	Expertise
6. Build integrity in	Perceived integrity
	Conceptual integrity
	Refactoring
	Testing
7. See the whole	Measurements
	Contracts

Much like in Scrum and XP, the relationship to quality improvement methods, and the drive for better quality is evident. In general, the agile and lean methodologies stem from the principle of continuous improvement (cf. Anderson et al. (1994); Brunet (2000)). Lean principles call for *situation awareness*; intrinsic understanding of the processes being performed, and the ability to alter them quickly to gain benefits faster. The principles represent a collection of "common management wisdom", and the aim is to reduce *waste*. In lean methodologies, this concept is central: waste consists of *partially completed work*, *extra processes*, *extra features*, *task switching*, and *defects*. In Kanban, waste is addressed with a set of high-level work management principles: *visualization of the workflow*, *limiting the work in progress*, *measuring and managing the flow*, *making process policies explicit*, and *collaborative improvement*.

Kanban techniques emphasize a technique of visualizing the workflow. This is achieved simply by using a signboard, or similar: The board has columns representing the work process, such as "To-Do", "In Progress", and "Done". Definition of done of a security-related task typically would include passing relevant tests and verification steps. Work items are moved sequentially through the columns, with the "in progress" column item limited to a fixed number of concurrent items. Lean methods were developed to address resource optimization and throughput maximization in mass production. Kanban addresses this mode of production as 'push' mode, whereas software is typically produced in 'pull' mode. In lean software process management, the waste is not limited to the limitation of tangible production items, such as items on the sprint backlog when using a Scrum-Kanban hybrid: It involves anything that is not directly producing value in the next release, according to Anderson et al. (2012).

Kanban and agile methods in general regard managing and leading the people as an even more important task than managing the work. The work is often highly complex, requiring a great amount of skill to be completed. To ensure continuous performance, Extreme programming explicitly lists sustainability practices, such as 40-hour week; Kanban abstracts this into "respecting the people". From the management perspective, these approaches are essential to keep the team performance level constant and predictable. This is also the basis for self-organization, letting the teams choose their tasks and give their own expert estimates on their complexity and the resources required. The respect for people also affects the quality control, as all raised concerns are to be taken seriously and the necessary changes applied quickly.

The individual improvements introduced by Kanban can be subtle. The principle is to introduce one-day changes: small incremental improvements to the work process that can be implemented in one work day. The focus is on quality improvement. Ideally, when a problem is found, the work

is stopped immediately to solve it: This serves the purpose of getting the quality right from the start, as defects are waste. This is an important aspect in software security engineering and would be reflected in for example the test-driven development practice of writing a test case for each found bug (Howard and Lipner (2006); Beck (2000)).

Besides a proof of concept created by Dorca et al. (2016) there is little extant literature about reported cases, or other examples of lean methods being used for software security engineering management. This can be hypothesized to represent a publication bias (i.e., preference not to submit scientific articles), rather than a lack of practical use.

3.1.4 Scaled Agile Methods

Scaled methods, such as Large Scale Scrum (LeSS)[1], Nexus[2] and SAFe[3] primarily enable larger organizations and enterprises to apply agile methods in their development work on the team level. They also affect the development process by adding their own inter-team communication practices and processes into the workflow of the agile practitioners. These methods provide the means to divide the product backlog to different teams and synchronize their work and output; more importantly, they promote the agile and lean processes through the whole management structure, not only the development teams.

These methods also provide a way to ease the larger enterprises into the iterative workflows without forcing them to drastically restructure their internal organizations. Much of the old management staff can retain their positions, allowing the bulk of the multi-level business organization to remain, and change only the interface towards agile development processes. In the simplest case, scaled methods such as Nexus allow large development projects to be split into small-size teams to successfully apply the agile development methodologies.

Scaling agile methods up involves managing distributed teams and controlling the organization overhead and a flexible workflow according to agile principles. There are some inevitable compromises to these principles, however: for example, the Disciplined Agile Delivery model by Ambler and Lines (2012b) approaches the management of agile production from the viewpoint of architectural control. It also divides the development process into specific management phases. This model may be suitable for certain delivery models, but when results – such as security fixes – are needed quickly, much of the "waterfall legacy" needs to be overcome if this logic is applied. On contrast, achieving the higher maturity levels in SSE-CMM or SAMM may

[1] https://less.works/

[2] https://www.scrum.org/resources/scaling-scrum

[3] http://www.scaledagileframework.com/

be inherently easier when using a planned work management and centrally managed organization.

A somewhat different approach to management of distributed teams and shared resources has been presented by Kniberg and Ivarsson (2012), by introducing a method commonly called the Spotify model. Besides the technical management of the "squads" (development teams with up to 8 members) work items, organizational and personal issues are brought into the focus by maintaining a measurement chart of the teams' performance and trends. This enables the management to take action based on the squads' direct feedback, and even predict future performance issues. Squads are divided into "tribes", which in the original concept were seen as mini start-ups. Tribes optimally consist of a maximum of 100 people, and exist to promote the social relationships within the tribe. Tribe can also be formed by the squads in one geographic location or building. In the Spotify model, squads retain the freedom to organize their own work and choose their preferred work management techniques, such as Scrum or Kanban.

In the Spotify model, technical work is managed by System Owners and chief architects. Information sharing between squads and tribes is enabled by introducing "chapters" and "guilds". Chapters are local to a tribe and connect the people sharing similar roles or technical task areas in different squads. Chapters have a technical lead, who controls the work profiles and information sharing within the technical area. Guilds are larger and looser entities, which may extend between the tribes and throughout the organization. Guilds enable people from various chapters to share common interests, goals, and methods, beyond their individual squads or tribes. A security guild, for example, could consist of interested programmers, testers, product owners, and managers.

The final examined scaled model is the Scaled Agile Framework (SAFe) by Knaster and Leffingwell (2018). From the viewpoint of software development, SAFe consists of production units, such as Scrum teams, working in iterations and creating products incrementally. The scaled framework is built upon a layered concept. Above the team layer, there are introduced the program layer, the large solution layer, and the portfolio layer. The development teams' work is divided into program increments, which provide workflow limits on the program level; in SAFe the continuous delivery pipeline is managed by these program increments, creating "agile release trains", scheduled work packages that contain several iterations' worth of planned work towards a release taking place when the release criteria is met. The upper layers in SAFe provide mechanisms to combine several trains, and add e.g. budgeting, and other necessary corporate management elements. Through this elaborate framework, SAFe also provides the necessary framework for security management, especially in conveying the contents of the organizational normative framework into the security requirements and the

release criteria.

The scaled models have quickly risen in popularity and use recently. In the State of Agile annual reports, SAFe has been the most popular scaled model since 2016; the Disciplined Agile model has also seen a considerable rise, from breaching the 1% threshold in 2016, to 5% in 2018. As the scaled models mostly address management issues outside the direct workflow of the security engineering processes, very little software security research on the topic of scaled agile models exists. An early work by Fitzgerald et al. (2013) describes an in-house scaling method, developed to extend the software team to accommodate the domain experts as well as the experts responsible for software safety and security. While this is essential for the security work, this type of scaling does not address the inter-team scaling issues. So far, the scaling models described in this section have not received much attention in software security research.

3.2 Security Engineering in Software Development

Security implementation in a software project is dependent on the normative, organizational and technological contexts, as defined by the ISO standard for Application Security, ISO/IEC Sstandard 27034-1:2011 (2011). Understanding these contexts is necessary to recognize the security rationale, and from them, the security objectives. The objectives state the security requirements. Changes to any of the contexts during software development is a source of security risk. This introduces the following challenges to the development of software security:

- Correctness, consistency, and completeness of security requirements
- Correctness, consistency, and completeness of security risk process
- Correctness, consistency, and completeness of implementation and verification of the security features and functionality

Defining the security requirements and identifying security risks is independent of the development methodology; implementation and verification of the security features and functionality is all about it. Achieving correctness in any of these fields conceivably benefits from applying an iterative and incremental approach, albeit the focus in this section is univocally on the security implementation and verification activities in the software development, analyzed against the three security contexts. The normative context, i.e., security regulation such as standards, guidelines, laws and directives, is the groundwork for security requirements. The normative context may

set requirements for both the development process and the software product, accompanied by requirements for security assurance. This also affects the choice of requirements engineering techniques, as shown by Elahi et al. (2011).

Security requirement engineering in an agile context may be performed through misuse cases first presented by Sindre and Opdahl (2005), applying SEI's SQUARE, or using mechanisms presented by the SSDLCs, evaluated by e.g. Tøndel et al. (2008). In an agile implementation of requirements, the typical mechanism is to first create user stories, or rather abuser stories which are then translated into items on the product backlog. At this point, a recommended approach is to apply formal quality criteria before accepting the items. Examples of such criteria are INVEST and SMART by Wake (2003): a backlog item must be Independent, Negotiable, Valuable, Estimable, Small, and Testable. When taken under implementation, a task should be Specific, Measurable, Achievable, Relevant, and Time-boxed.

After implementation, agreed and discrete Definition of Done criteria are to be applied, based on the requirement elicitation technique. The greatest difference between adaptive and prescriptive methods lies here: agile development consists of piecemeal definitions of the requirements, rather than doing everything in advance and at once. This allows not only more accurate work load estimation but also much greater technical precision, improving security.

3.2.1 Security Requirements Engineering in Software Development

An agile security development process, when following the agile principles, should be utilized to define and implement only the necessary security and security assurance. To reach this goal, the correct setting of security rationale is of critical importance; this cannot be achieved without the bottom-up risk management process, identifying the protected assets and risks. Normative frameworks set the rest of the objectives.

In an agile mindset, the security rationale is based on a dynamic evaluation. Extending upon the agile principles and terminology, the result of this evaluation can be called finding the *Minimum Viable Security* (MVS). The principles to achieve MVS can be directly derived from the main challenges to software security:

- Security objectives are correctly, completely and consistently defined;
- security objectives are correctly, completely and consistently met; and,
- sufficient security assurance is provided.

Agile development methods provide an adaptive framework for the build process. Formally this is realized by the attitude towards changes in the requirements set for the development. In the agile framework, as the iterative work progresses, any number of incorrectly set requirements may have to be redefined, while "new", i.e., latent, requirements are discovered along with the development work, as knowledge is accumulated. By minimizing the assumptions and defining the requirements as they appear, the resulting set should result in a minimal set of requirements to achieve the security objectives.

The standards guiding software requirement engineering have been reviewed by Schneider and Berenbach (2013), providing guidance and criticism also applicable for security requirement elicitation. A comprehensive survey by Tøndel et al. (2008) lists several approaches and techniques to perform security requirements engineering: The analyzed techniques typically combine risk engineering with threat modeling, or security design. To put this into a formal framework, these methods represent the knowledge base for the evaluation process. The evaluation process sets concrete metrics and formal criteria for the systems' build process, as well as their results.

Security requirement management is largely based on the security risks identified during the development. Risk identification, and consequently the requirement definition, demand considerable security knowledge and awareness of the security issues from the developers. In modern software development this knowledge is much supplemented by the use of automated tools: when applied in the technological context, they enable the software and security engineers to concentrate on the correctness of the security process and the assurance produced, rather than the technical details.

The compliance of the development process should always be secondary to the compliance of the software product under development, unless the work is performed under specific compliance requirements. The requirements for the process should be selected case by case, implemented and applied to meet the actual security objectives and to provide true security. The security objectives and the security criticality of the software under development may also influence the software development processes; most typically in literature, a security expert role is added to the development team, taking responsibility for the security work and coordinating the security objectives (see Howard and Lipner (2006)).

The foremost shortcoming of the SSDLCs presented in the previous section is the inadequate definition of requirement engineering. Formal requirement engineering methods such as Security QUAlity Requirements Engineering (SQUARE), by Mead and Stehney (2005), amended by Mead (2015), aim to provide measurability and rigour to this process. These methods provide means to both defining the security requirements, and identifying security risks. SQUARE suffers from the same general shortcoming as the SSE-

CMM, and other prescriptive management-heavy methods: While thorough and comprehensive, they scale poorly for smaller organizations. Additionally, they impose severe resource and schedule strains on the software development process, causing implementation difficulties in also larger organizations. As such, these methods are primarily useful in providing guidelines for the processes. The implemented engineering process will most likely be a subset of the suggested processes, always adapted to the organization and the current objectives.

3.2.2 Security Risk Engineering in Software Development

ISO/IEC has defined a standardized risk management framework for information security in ISO/IEC Standard 27005:2018 (2018). The ISO standard differs somewhat from e.g. the Touchpoints risk model, or even the SSE-CMM model, in both terminology and the level of details. In an adaptation of the risk management process for government work by Patiño et al. (2018), the risk management process is performed in four distinct stages. After the risk management context has been established, the process progresses bottom-up: each component subject to the process is evaluated individually, and the combined risk is then evaluated.

Process stage 1, Context establishment, contains three activities:

- Scope establishment. Define the environment and critical services.
- Selection of critical processes. Includes involving the people and managing the required resources.
- Description of Evaluation Criteria. Qualitative risk estimation.

An example result of this phase could be a risk matrix, used as a template into which each critical process is placed. The matrix has two dimensions: Frequency and Impact. The total risk is the product of these two. The frequency varies from highly improbable (1) to highly probable (5), and the impact from very low (1) to very high (5).

Stage 2, Risk identification, contains five activities:

- Identification of assets – in software development, an architectural description or security-related use case list can be utilized.
- Appraisal of critical assets. Estimate the impact of each asset included in the process. Calculate the average to gain an overall asset impact value for the process.
- Identification of threats. This activity is based on security expertise and knowledge of the threats.

- Identification of controls. Create a checklist of controls to mitigate the threats, update the list as necessary.
- Identification of vulnerabilities. These may vary from natural to social and technical and must be monitored.

The work products of this stage are a list of critical assets, a list of threats, a list of controls and a list of vulnerabilities. When the impact of the risk to assets is appraised, Patiño et al. (2018) suggest that processes with an average above 3.00 are considered critical. In this stage, the significance of security awareness is heightened. Identification of threats, controls, and vulnerabilities are highly context-specific and require an adequate level of security expertise.

Stage 3, Risk Estimation, consists of two activities; these are performed for each item on the lists created at stage 2.

- Valuation of probability. A numeric value from 1 to 5.
- Valuation of impact. A numeric value from 1 to 5.

Assigning the numeric values requires expertise on both the business domain, and the security vulnerabilities and their impact. The standard does not give any advice as to how to perform this critical phase: It is important to know that all the related risk components are correctly identified and that the appraisal of the risk and its impact is accurate. To make things worse, the environment and thus the probability and impact of the risk may be in a constant state of change. To manage the ongoing risk assessment process, an owner should be designated for each risk. The owner may not be responsible for the mitigation or even remediation of the risk but can be held explicitly accountable for the management of their particular risk items. Risk can be expressed the direct product of the probability and impact: ($Risk = Probability \times Impact$), properly adjusted to the asset value. Security controls are implemented to reduce this value.

An example risk analyzing technique, called after its mnemonic DREAD, combines this stage with the next one, Risk Evaluation. The mnemonic comes from *D*amage potential, *R*eproducibility, *E*xploitability, *A*ffected users, and the exploit's *D*iscoverability. This model adds further dimensions to the risk estimation process, and makes the impact assessment more thorough by taking e.g. the amount of affected users into consideration. DREAD has been integrated also to tools, as described e.g. by Ingalsbe et al. (2008); as Chapple et al. (2018) notes, these inspection points should be accompanied by risk estimation aspects customized to the current context.

Stage 4, Risk Evaluation, concludes the risk process and consists of three activities.

- Risk valuation. Calculate the product of each impact and frequency. To be repeated once in e.g. 6 months.
- Identification of critical risks. Focus on mitigation of risks that are ranked high or very high.
- Selection of control measures. Treatment options are: mitigate, accept, avoid and transfer to a third party.

This stage sets the requirements for the practical security engineering work. In a software development company, the risks that are to be mitigated or avoided require policies and guidelines. These policies and guidelines form a major part of the organizational normative framework. Through application normative frameworks derived from them, they get incorporated into the software development processes, environments. They are also implemented into the software products. The risk management framework pays specific attention to an organization's internal threats. Internal threats, also known as malicious insiders, were also a major concern of the early security standards: the military did not trust their civilian contractors.

To avoid excess security overhead the security risk process should be a part of organizations generic software risk process, such as one described by Boehm (1991). The process described by Boehm is not specific to software security, but security items can be easily added to the to it. This process consists of six steps: (1) Risk-identification checklists, with a top 10 risk sources (more, when security is included) and a quantification model for this; (2) Risk analysis and (3) risk prioritization are to be adjusted to the security items relevant to the current context; (4) Risk-management planning, involves mitigation for the risk items is integrated; (5) Risk resolution and, finally (6) risk monitoring, involve implementing the plan and by keeping the implementation on track by a closed-loop process and applying corrective action whenever necessary.

Risk management sets the foundation for all the security work in the organization, by practically defining the organizational security rationale. Combined with the security regulations and other external factors, it forms the complete Organizational Normative Framework, from which each Application Normative Framework is derived. An example of such external regulation is a governmental requirement for use of a risk management framework or a security maturity model, such as NIST RMF (NIST 800-37) in the United States, or the VAHTI framework in Finland. The caveats of quantification of risk probability are obvious: Such calculations should be used with caution, especially when used to manage security-related investments and security work.

An essential characteristic of the risk management process is that it cannot be done by the developers alone: input from the user domain is

essential for the completeness of the assessment. Perhaps due to this, and the amount of work involved in the risk process, the reported experience of risk management in software security research is scarce. This indicates a research gap and a clear demand for improved risk management processes. One such model could be a combination of iterative risk process keeping track of the top 10 (or any number found suitable) of the software security risks in the development project, and managing the security engineering and assurance work utilizing this list.

3.2.3 Security Assurance Engineering

Security assurance has many definitions, and its scope and composition has significantly grown since the introduction of the concept. The assurance elements described in the first DoD standards still form the basis of software assurance in software-intensive systems. In the early definitions, assurance consists of system design and security documentation, together with descriptions of security policies (cf. DoD (1985b)).

Assurance in the security context is commonly specified as *grounds for confidence that a deliverable meets its security objectives.* In ISO/IEC Standard 15026-2:2011 (2011), a definition of assurance is given as a set of *evidence* consisting of the sets of known *facts*, *data*, *objects*, *claims* and *assurance cases*. Of these, a claim is further specified as a proposition to be assured about the system of concern, chosen based on a set of justifications for the objectives. Software (security) assurance is defined as "level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle and that the software functions in the intended manner".

Hamidovic (2012) combines security assurance engineering with requirements engineering in his practical dissection of security assurance selection and implementation. This approach to security assurance reflects the three main areas of software security engineering:

1. Security of the deliverable software through verification and testing.
2. Security of the development processes.
3. Security of the operating environment.

Security assurance exists to create confidence in all three. In an analysis of information assurance techniques by Such et al. (2016), four main components of assurance are identified: (1) Assurance scheme, consisting of standards (e.g., ISO 27001) and qualifications; (2) Assurance target, which are further classified as either security controls or security competence required for assessment; (3) Assurance technique, the method of assessing the

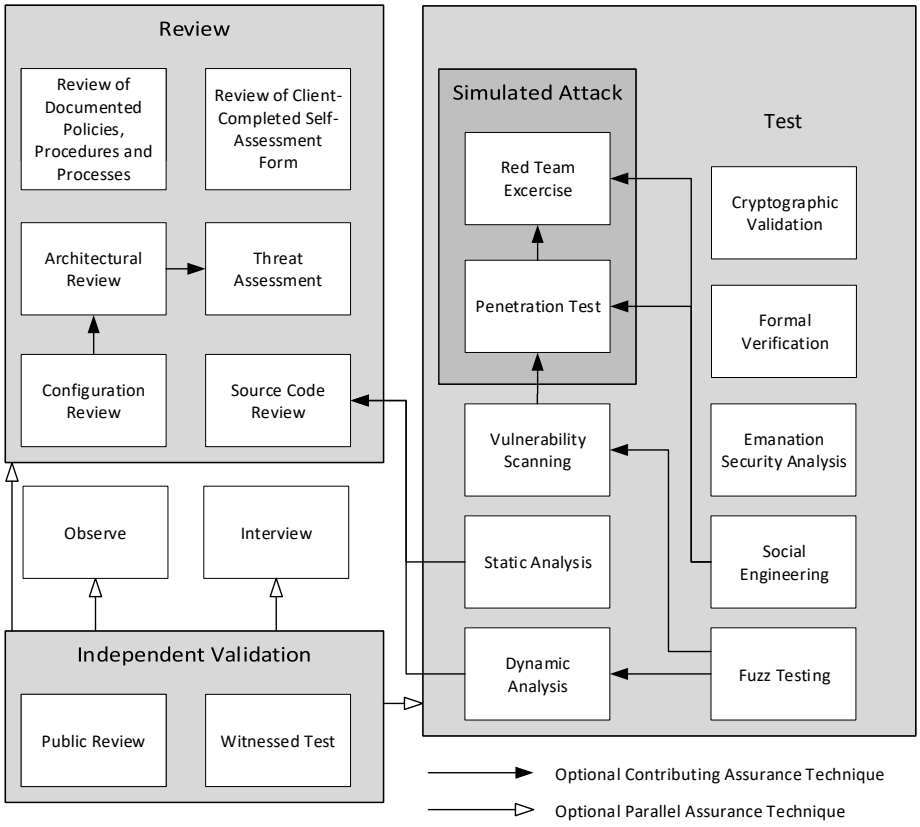


Figure 3.3: Assurance techniques as given by Such et al. (2016)

target: either security control, such as penetration testing, or controls for the qualifications of the personnel; (4) auditing and assessment evidence of the three other components. All four of these are directly applicable to security context as well. The assurance techniques identified in this study are presented in Figure 3.3.

Not all of the assurance techniques are directly used by the observed software security assurance frameworks, but they may be introduced to the assurance process by e.g. regulation. The level, scope and qualities of security assurance are ultimately dependent on the applied security policies. Beznosov and Kruchten (2004) group security assurance into *requirement assurance*, *design assurance* and *implementation assurance*.

Implementation assurance is gained by security reviews and various forms of security testing. In a cross-case analysis, Cruzes et al. (2017) identify the following risk-based security testing types in various software development life cycle phases:

- Design: Model-based Security Testing.

- Implementation: Code-based Testing and Static Analysis
- Verification: Penetration Testing and Dynamic Analysis
- Maintenance: Security Regression Testing.

To set the target of the validation's outcome, the assurance argument must be clarified. In defining the minimum viable security assurance, the NASA definition of minimum security assurance as listed by Davis (2005) could prove helpful:

1. Security risk evaluation process has been performed.
2. Security requirements have been established for the software and data.
3. All software reviews and audits include evaluation of security requirements.
4. Configuration, change and incident management processes contain security reviews to prevent security violations.
5. Physical security for the software and the data is adequate

In addition to the proof of secure system architecture and design, the security assurance requirement includes verification and validation of appropriate security controls. This assurance is created to provide an acceptable amount of information about the security policies being enforced. In practice is created by logging the system and application events. The core of security policies is to set the rules for how the access is granted to the data contained in the system. The access control in based on the authentication scheme: the security classification of information and its users.

3.2.4 Security in Continuous Delivery

Depending on the organization and the software product, the iterative and incremental software development have extended the practice of *continuous integration* into *continuous delivery*. The merger of development and operations is called by the portmanteau *DevOps*. In continuous delivery models with a special emphasis on software security, the model is sometimes called DevSecOps, or some of its variants (cf. Mohan and Othmane (2016)).

Organizations apply the continuous delivery models and processes to speed up the software deployment, by creating a production pipeline with a high degree of automatization. This level of streamlining calls for good quality code and extensive and highly automatized testing processes, as described by Kim et al. (2016). For software security, this poses both new challenges and opportunities: when a service-type software is published in

the open Internet in a matter of hours, or minutes, the security testing and monitoring tools need to be constantly up to date and able to respond to changes. The changes may occur in the software or its configuration, or in the operating environment in the form of new security threats or even infrastructure changes (Mohan and Othmane (2016)).

The change does not happen only by introducing tools and automatization: it requires also change in the organization, bringing the developers and operations closer to each other, often over the subject of testing (Kim et al. (2016)). Also in regulated environments, such as security-critical ones reported by Mohan and Othmane (2016), or safety-critical by Laukkarinen et al. (2018), this results in an increase in effectiveness and flexibility and a decrease in response times. This fully fits the definition of the word "agile", whether used as an adjective or noun.

Continuous delivery impacts security in the form of requirements for scanning tools, security monitoring, and security logging. The work management tools will also need to be able to accommodate and appropriately prioritize the security-related tasks created from the operations and inserted into the development backlogs as discussed by Mohan and Othmane (2016). For example, Kanban-style work management should allow such flexibility, whereas a scaled agile model, such as SAFe with release plans, may face greater challenges in pushing these changes into production. Moyon et al. (2018) provide a direct example of how to modify SAFe to achieve compliance with ISO/IEC security standards. Proper prioritization of the security items is one of the most pressing issues in software security research and practice.

The security assurance engineering is also facing changes in a continuous delivery model. Security assurance may take new forms, such as archiving entire virtualized environments and using them as security evidence (cf. Laukkarinen et al. (2018)). After an incident, they can be used as a source of security forensics. Ideally, technical security assurance consists of elements usable not only in security engineering but also by the software's developers and operators.

Security Management in Software Development

In a standard-regulated security scheme, the security objectives are normative and formalized. Objectives are translated into requirements and policies, which in turn are implemented by security functionality and verified by security assurance.

Boehm and Turner (2003a) proposed the choice of software development method to take place on an axis between Agile and Plan-driven methods. He proceeds in presenting five dimensions for this selection. Presented roughly from the most quantifiable to the most qualitative, the dimensions are:

1. Size, the number of personnel;
2. Dynamism, or number of requirement changes per month;
3. Criticality, or amount and severity of loss due to impact of defects;
4. Skill of the personnel, as defined by Cockburn (2002) and further developed by Boehm and Turner (2003a);
5. Culture, ranging from "chaos-thriving" to "order-thriving".

The agile methods evolved to meet the shortcomings of the prescriptive methods, in conformance with the best practices and management values in software engineering. After two decades, the simplest and most flexible methodologies appear to have thrived: Scrum, XP and their hybrid forms still have a strong representation in the industry. A notable trend has been the emergence of various *scaled* agile frameworks, such as Large Scale Scrum (LeSS), Nexus and the Scaled Agile Framework (SAFe), providing management models and practices for several coordinated development teams, thus enabling the agile methods to be used in larger organizations.

The developments in agile methodologies have amended the juxtapositions between agile and non-agile methods, described by Boehm and Turner (2003a): scaled methods relax the team size limits, and there are methods introducing a level of pre-planning into the iterative and incremental development process, while still claiming to retain agile principles. Examples of such methodologies would be Disciplined Agile Development (DAD) by Ambler and Lines (2012a), or the slightly older Crystal family by Cockburn (2000). Neither of these has acquired significant reported use in the industry, but contain elements useful for producing more secure software through their focus on architecture and requirements management. Boehm and Turner (2003a) also presented the concept of "home grounds" for agile methods, another concept that has been outdated by the methodological developments and at least partially later predicted by Boehm (2006) himself.

At enterprises the use of scaled methods appears to be dominated by various "methodology providers": a true competition of agile or lean management ecosystems, complete with extensive training models and certification programs. In this competitive landscape, Scrum has fared best, despite being divided into factions of scrum.org[4] and Scrum alliance[5]. As shown in Figure 3.1, among the most popular agile methods are only a few established methodologies. The methods are being morphed into various hybrid models and scaled up into more elaborate product management models.

[4]http://www.scrum.org
[5]https://www.scrumalliance.org/

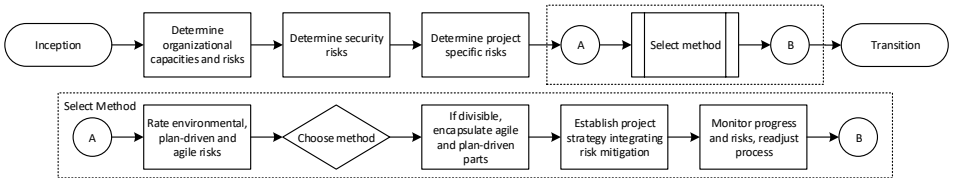


Figure 3.4: Decision-making in risk-driven development method selection (adapted from Boehm and Turner (2004))

Despite the near-absolute current dominance of the agile methods, not all projects may warrant using them. While agile methods dominate, there may still be organizations or situations where use of "non-agile" patterns may be justified to achieve the optimal target; markedly, these include projects with fixed schedules and budgets. In a critique to agile methods, Boehm and Turner (2003b) discussed the dimensions the selection process is based on. Boehm and Turner (2004) formalize this approach into a simple decision process, illustrated in Figure 3.4 and set into the context of risk management.

In the figure, the risk evaluation process is used to give the necessary insight to the validity of the security objectives, and to help determine the appropriate security assurance. The decisions should also be based on the resources at disposal: skill, tools, workforce and time. Especially the amount of skill is a critical factor in security work; requirement elicitation, risk identification and the mitigation techniques all require extensive knowledge in the security field. The significance of the individual is markedly emphasized in agile software development, coiling back to the significance of skill. A framework for formalizing skill levels in software development has been presented in Cockburn (2006); from the software security perspective, the skill requirement stresses the importance of relevant security training.

In addition to the security processes, also the agility of a process may be a subject of evaluation. Schweigert et al. (2014) identified about 40 models claiming to be agile maturity models. They pointed out significant difficulties in defining the agile processes and metrics in a way that would be commensurable between the agile maturity models. The agile models deal with general organizational processes and are directly applicable to security engineering. To separate fads and buzzwords from solid engineering principles, Séguin et al. (2012) performed an analysis of agile principles. The bulk of them was found to be valid software engineering principles. These findings from both practical engineering and research traditions support the use of agile methods and principles for purposes beyond managing singular projects.

Security objectives are drawn from the environment and the protection target; security objectives are met by setting security policies; these form the concrete security requirements, elicited by a risk-driven process; the security

requirements are further developed into verifiable features and functionality. Security assurance largely 'emerges' from this process, but may also require formal reviews, security audits and few pieces of security documentation, most importantly a security architecture. The depth of security assurance depends on the objective, but should always exist to serve the purpose of increasing security and facilitating security management.

Lean methods can be used as a Software Process Improvement method far more flexible and effective than traditional maturity models, as shown by Poth et al. (2018). Heavily biased towards solving the technical and practical, rather than the managerial issues, it can also be an effective way to manage the realization of the security rationale, and production of the appropriate security assurance. According to lean principles, this involves abolishing hierarchical organization, directly managing the people and team dynamics, addressing the faced challenges immediately and focusing on quality. Combined with diligent management of the process and project, based on direct feedback from developers and customers, the lean principles claim to enable the organization to always work on the most valuable item (see Poth et al. (2018) and Poppendieck and Poppendieck (2003)).

3.3 Summary

In software development, security is a collective term, and a means for achieving software reliability, confidentiality, integrity, availability, and compliance. Security engineering has direct convergence with those software engineering practices that traditionally have been used to improve software quality. The agile methods aim to the reduction of unessential work, and the improvement of quality. Security engineering, in addition to being a requirement in an increasing number of cases, heavily contributes to the latter. The main mutual contribution of combining the security engineering techniques with effective software development practices is specifically the increased rigor in requirement and risk engineering, and enhancements in the quality assurance techniques.

Excess security compliance requirements may entitle allocating resources towards administrative security. In software development, this is realized in the form of additional security assurance requirements. In an agile or lean process, additional verification-based requirements create potential bottlenecks that need to be addressed. In these cases, instead of being the driving force for e.g. better testing coverage, or more thorough code quality checks during security reviews, security processes may become a hindrance to productivity. Proper inclusion of security processes into all development stages removes these bottlenecks. Conversely, agile methods help in the validation of security, as the iterative process with tight requirement management

practices has been designed to "build the right product" from the start, including security requirements.

Management of security engineering work in an agile software development project is a combination of various techniques. These are to be fitted into the iterative and incremental framework and rigorously applied according to the lean or agile principles. Arguably, any software project would benefit from applying even a light-weight security requirement elicitation process, identifying e.g. 10 security risks and implementing a management plan for those risks; practical programming guides, such as OWASP Top 10 security flaws should also be put to good use in the implementation phase. Also, one of the project's aims should be the production of appropriate security assurance: Optimally, assurance-producing mechanisms also double as forensic evidence after the occurrence of a security incident. To systematize the approach to software security, a maturity model, such as SAMM or a light-weight version of the SSE-CMM, could be modified to fit to the organizations' purposes, providing an ample target for future research.

Chapter 4

Research Description

The research objective was to find practical solutions and challenges in production of secure software. For relevance, the chosen methodologies and the research results reflect the current best practices and state of the art. The steps to ensure both scientific and technical relevance were taken in the form of literature surveys, case study and an industry survey, following the frameworks described by Wohlin et al. (2012). Conceptual models and analytic frameworks were utilized to gain an understanding of the topic and to provide guidance in the empirical studies.

In this chapter, the research structure is presented by describing the individual research articles included in the research. The research methods, key contributions as well as possible limitations and openings for future research are presented for each article.

4.1 Research Structure

The research consists of six publications, utilizing a variety of research methods to achieve the research objectives, and were selected to provide a thorough view of the research field and research topics. The publications, and the primary research methods used, are given in Table 4.1.

Table 4.1: Publications and the primary research methods

Publication	Chapter	Research Method
P1	4.2	Literature Review
P2	4.3	Constructive Research
P3	4.4	Constructive Research
P4	4.5	Literature Review
P5	4.6	Case Study
P6	4.7	Survey Research

The field of research was charted by reviewing the relevant literature; constructive research was used to delve deeper into key aspects, further clarified by additional literature review. The findings were validated by a case study and survey research.

Publication P1 provides a review of literature in the subject of agile security engineering, gathering further empirical evidence of how security mechanisms are applied to agile software development methods. The material for the study was gathered from five online libraries, from which 11 peer-reviewed articles were selected for analysis. The selected articles contained empirical cases of applied agile security engineering. The security mechanisms in each article were identified, and a security activity matrix was compiled based on the findings. It contributes towards finding answers for RQ1, defining the security objectives and rationale.

Publication P2 presents a framework of mechanisms to set and meet the security requirements and objectives in agile software development. The framework is based on secure software development life cycle models and their application to the agile software development process based on extant literature and security standards. The development techniques involve security objective setting, requirement elicitation, and the design, implementation and verification of security features and functionality in the agile software development process. In addition to the contribution for RQ1, in this publication, the currently prevalent agile software development practices and software security engineering are thoroughly surveyed, solidifying the practices to combine them. This knowledge is used to answer RQ2, defining the concrete software security engineering methods.

Publication P3 further develops on the research topic of meeting security assurance requirements with an agile software development process: in this article the most widely used software development method, Scrum, is enhanced to meet the security assurance requirements of the VAHTI instructions for the software development processes. The resulting VAHTI-Scrum method is a software development method capable of meeting the VAHTI security requirements and producing the required security assurance artifacts. This article provides a contextual solution to RQ2.

Publication P4 is a comparison of key industry surveys in the field of software security engineering. BSIMM, a main software survey apparatus in the field of software security, is an annual international industrial survey conducted among companies with an established software security function. The results of this survey are then utilized in the BSIMM to construct a descriptive maturity model, with 12 core practices "everybody does". The results are directly compared to an industry survey conducted in Finland, pointing out considerable differences in the core practices, and revealing a severe lack of security training offered by software employers in Finland. The information is used to answer RQ2, and to map the research field and the

head topics of RQ3, the challenges and impact in practical software security engineering.

Publication P5 describes an industry case of applying the Scrum method to produce a VAHTI regulated software-intensive product, in this case, an identity management platform for high-security service infrastructure. The article provided supporting evidence for Research Questions 1 and 2, but its primary goal is to answer RQ3.

Publication P6 is an industry survey extending the empirical research. The survey was conducted by selecting security engineering practices from prevalent security development life cycle models, and combining that with extant industry surveys about agile software engineering. The aim of the survey was to attain knowledge of the current use of security engineering practices in the context of agile software development, and their perceived impact on security. The survey gives a detailed view on current SSE usage in the Finnish software industry and its perceived impact. The results are a direct contribution to RQ3.

4.2 P1: Busting a Myth: Review of Agile Security Engineering Methods

In this study, the history of security engineering processes and activities integration with agile software development methods was researched by a literature survey. The inception of iterative and incremental software development methods raised suspicions of an inherent incompatibility between the traditional non-agile security processes and the new agile methods. This suspicion is deemed to still affect the attitude towards the suitability of agile methods to provide security. To examine and explore this claim, this study presents a literature review of a selected set of agile secure software development methods.

The results show a wide and well-documented adaptation of security activities in agile software development, with the observed activities covering the whole security development lifecycle. During the research process, a repetitive theme was noted in the reviewed articles: it was noted that agile methods are considered unsuitable for security engineering work, not to mention to achieve regulatory compliance or conformance with a capability maturity model. However, the evidence to the contrary is excessive, and the reviewed articles provide many concrete examples of empirical cases where security engineering has been applied at all phases of the software security development life cycle in the context of agile development.

Table 4.2: Literature searches and result set size (truncated from P1)

Library	Result set size
ACM Digital Library	59
IEEE Explore	120
Springer Link	101
Wiley	32
ScienceDirect	64
Total	376

4.2.1 Method

A systematic literature review, following the principles set by Kitchenham et al. (2009), was used to find the definitive set of secure agile software development methods, of which a core set of 11 papers was selected for analysis, and the security activities documented in the methods were extracted. The research was conducted in several phases: first, a systematic literature method was applied by performing searches in five online computer science libraries, using a systematic and documented process. The resulting set of 388 articles was then manually screened for most suitable examples of the adaptation of agile methods to security work, with a preference for case studies and using the incorporation of security engineering activities as a selection requirement. A simple metric, citation count, was considered an indication of the impact of the research. Due to inconsistencies in citation count reporting between online libraries, a high number of citations was not an essential requirement, but merely a contributing factor towards inclusion.

In Table 4.2 is shown the amount of the agile software security engineering articles at the time of the article's writing, in the selected online libraries.

Each library search was conducted with the term `(agile AND software AND (security AND engineering))`, with slight library-dependent variation in syntax. This body of articles was used to select a representative set of software security engineering research, covering the whole life cycle. The selected articles were then reviewed and analyzed, and the security activities described in the articles were identified. These activities were evaluated using the principles of Common Criteria, which is the ISO standard to "permit comparability between the results of independent security evaluations". It does so by providing a common set of requirements for the security functions of IT products and systems and for assurance measures applied to them during a security evaluation.

Finally, each method was screened for any empirical evidence provided, as were any references to security, quality or safety standards. The results were categorized and evaluated using the defined analytic lenses.

4.2.2 Contribution and future work

Based on the findings of this study, the practice of security engineering appears well adapted to, and widely used with, the agile software development methods. Some of the activities have been modified to better suit an iterative development model and, based on the literature, much attention has been paid to retain the agile nature of the development process despite of the added security activities. The often-repeated myth of agile methods' incompatibility or inherent unsuitability for security tasks and achieving security objectives is still being perpetuated.

Selected result articles represent the various phases in the SSDLC models thoroughly. Notably, a minority (4 of 11) were empirical. The impact of these articles was not reliably available, so the direct number of references was used as reported by the library itself; this was acknowledged not to be the actual number of references. The results show the history of the agile software security engineering research, as well as its overall typology: conceptual papers are dominant, with empirical evidence provided mostly in the form of case studies. The methodological development of the agile software development processes is a central theme, complimented with the research of security skills and management.

The field of secure agile software development is still fragmented and organization specific, largely due to the highly adaptable nature of the agile methods. A wide industry survey concentrating on agile security activities would help to identify the key agile security practices and confirm the findings outlined in this study. This was performed in P6.

Also using other than peer-reviewed academic sources would provide an interesting ground for hypotheses, to be verified with a scientific method. Finally, distinct similarity between security and safety activities implies an increase in the general quality of the software products created with security-augmented processes. The impact of security activities on the measurable quality of software, and software project management should also prove to be an interesting research subject.

4.3 P2: Aligning Security Objectives with Agile Software Development

The research objective of this article was to provide a mapping between security engineering and software engineering practices, and to clarify the transformation of security objectives into verifiable security functionality and security assurance, utilizing a measurably agile software development process. The work is based on security practices from several software security engineering frameworks and derives the agile practices from industry

surveys. Development-time security practices were derived from the Microsoft SDL, the ISO Common Criteria, and OWASP SAMM. The resulting framework is inclusive to various types and levels of security requirements, and can be used to guide the selection of both software development and software security engineering practices in a software development project.

4.3.1 Method

The research framework was a literature-based constructive model built on the software security development life cycle models. The security engineering practices in each phase of the chosen models were listed, and agile practices supporting the activity were found. A mapping between these practices was performed, resulting in a framework comprising of both agile and security engineering practices.

The relevance of the framework was increased by selecting the agile practices from industry surveys, and the security practices from industry life cycle models, maturity models and ISO security standards. The model to map the software security engineering practices into agile software development was derived from extant literature documenting the use of security engineering in software development. The resulting was created as a combination of empirical and conceptual work.

4.3.2 Contribution and future work

The resulting framework provides useful guidance in the selection of a suitable agile framework for security engineering work, or for improvement of software security in an existing software process by introducing compatible engineering practices. The framework should also improve the efficiency in achieving the security objectives by integrating the security processes into the software development processes, instead of running them separately and therefore needlessly using resources. The key results are improvements both in the security of software, and the efficiency of the processes used to produce them. The resulting mapping is presented below in Figure 4.1.

Modeling an agile process is necessarily always an approximation, as the implementation of the process may significantly deviate from the model. The mapping does, however, show the primary execution points of the agile practices in the SSDLC. The framework prompts for empirical research. As such, the framework can act as a guideline for future work, not a strict selection criteria for industry use. Agile organizations and software development processes are flexible, so the usefulness of a singular unified model can be very limited. Instead, flexible and extensible frameworks can fit several organizations and objectives, proving the most utilizable end results. Another beneficial research approach would be to reverse the angle and enhance the

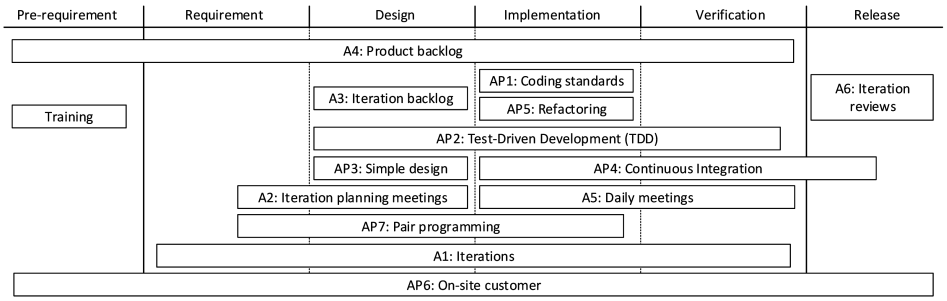


Figure 4.1: Agile activities mapped into the security development life cycle (from P2)

identified agile practices by software security engineering practices. The resulting framework, an enhanced practice set, could then be applied to any methodology the organization decides to use, to best achieve the current objectives.

4.4 P3: Securing Scrum for VAHTI

This article provides an insight into the combination of methods, techniques, and tools used to promote data confidentiality, integrity, usability, availability and privacy. Specifically, this is presented in the context of conducting an agile software development project that needs to comply with VAHTI security requirements. In order to achieve concrete and measurable levels of software security, several international, national and industry-level regulations have been established. The Finnish governmental security standard collection, VAHTI, is one of the most extensive examples of these standards.

The article presents a selection of methods, tools, techniques, and modifications to the Scrum software development method to achieve the levels of security compliant with VAHTI instructions for software development. These comprise of security-specific modifications and additions to Scrum roles, modifications to sprints, and inclusion of special hardening sprints and spikes to implement the security items in the product backlog. Security requirements are transformed into security stories, abuse cases and other security-related tasks. The security activities were divided into three classes based on VAHTI security levels. The security assurance artifacts were identified, as were the tasks to create them. The *definition of done* in software security is established to provide compliance with the VAHTI requirements, and the steps to achieve it are presented.

4.4.1 Method

A conceptual-analytic research approach was used, combined with a constructive research strategy (see Järvinen (2004)). As a practical research activity, this translated into an analysis of the integral elements of the Scrum framework and VAHTI security regulations: based on the conceptual analysis, the article proposes modifications to Scrum that it will fulfill the requirements set by VAHTI.

This study presents the result of conceptual analysis and modifies the Scrum process and framework to adapt to the security requirement. The presented model was based on a conceptual and analytic approach and it has not been empirically verified. While the two concepts were carefully examined, and the new model created based on the results of this examination.

4.4.2 Contribution and future work

The target of analysis is a modern security maturity model, VAHTI 1/2013, designed directly for software development. The maturity model also acts as a security standard, as it states clear security requirements for the process and its outcomes, without specifying the implementation itself. The security assurance requirements of VAHTI were enumerated and evaluated against an agile framework, which was then adapted to comply with the requirements and produce the regulative security assurance. This study rather pragmatically concentrated on the *security assurance* as the definition of done when aiming to compliance with a security regulation. To provide future software development projects with direct instructions to achieve compliance, a detailed diagram containing all the required security assurance artifacts was created. This diagram is presented in Figure 4.2.

The map presents the various levels of the maturity model (Basic, Increased, and High) and the type of the assurance artifact. Initial documentation should presumably exist before the development process starts, and development-time documentation is produced during the agile development in iterations and may be incrementally updated. External documentation is either a product of independent verification and validation processes or specific development-time documentation produced for the maintenance organization, historically separate from the development organization. In a continuous development model, the separation of duties does not necessarily exist, or at least it is not that strict; in this organization type, maintenance documentation is part of development-time assurance.

To continue on the paths opened by this research, the benefits and drawbacks introduced to security and safety development by agile methods should be further studied. The most important question is, how the items used to

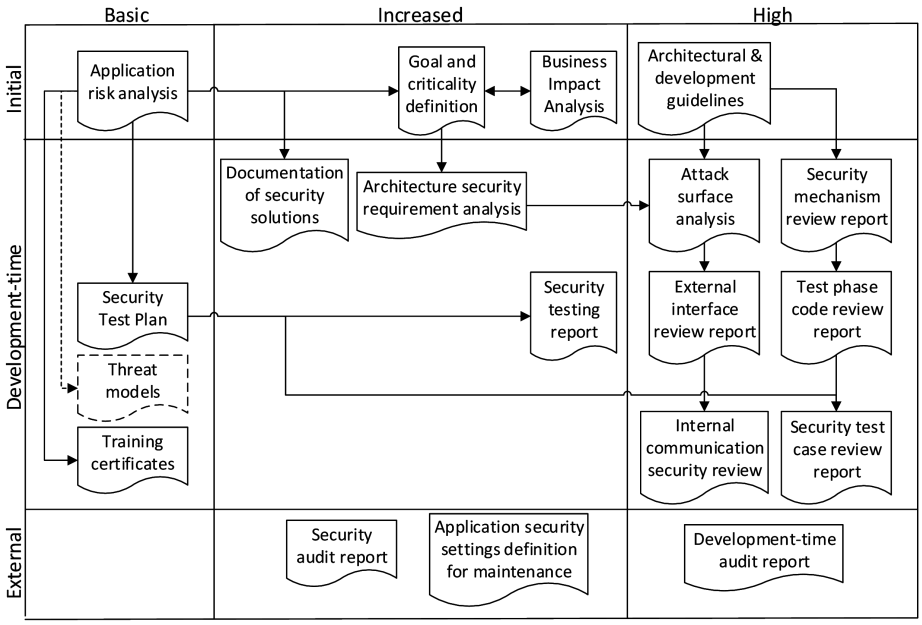


Figure 4.2: VAHTI documentation artifacts (from P3)

provide security assurance actually advance and compare with the actual security of the software product. Based on a conceptual study, it is difficult to perceive the actual degree of complexity and depth of security actions; the exacting of the security measures is performed by the development team, security experts, and finally verified by security auditors. Without an instantiated process and a case study, complete with external security audits, any in-depth evaluation cannot be possible.

4.5 P4: Surveying Secure Software Development Practices in Finland

In this article, the current state of security engineering surveys and results from an industrial survey in Finland are presented and compared. The main comparison is performed towards BSIMM version 8 and the similarities and distinct differences are discussed. The research objective was to compare the findings of the BSIMM to the data gathered in a Finnish survey, and to perform an analysis of the BSIMM survey using an evaluation criteria for industrial surveys used and presented by Stavru (2014).

An analysis of the composition of security development life cycle models is presented, suggesting regulation to be the driving force behind security engineering in the software industry. It was also observed that there is a

non-correlation between the frequency of usage and the observed impact of the software security engineering practices. Among the findings was also the observed lack of security training provided by the Finnish software companies: only 16% of the respondents reported having received security training from their current employer. In comparison, BSIMM's reported figure is nearly 70%. This can also indicate a security bias in BSIMM, as this study can be taken to represent security-oriented software development companies, while the Finnish survey was performed among the general population of software engineering practitioners. Taking into account that the answers were received mere weeks before the European Union's General Data Protection Regulation (GDPR) was to be enforced, the figure may be considered outright alarming.

4.5.1 Method

The research was conducted by a statistical analysis of certain aspects of the results of an industry survey, described and reported in P6. The survey asked software practitioners about their development practices regarding both software engineering and security engineering, and was performed among the SW professionals working in Finnish software companies. The objective was to utilize the gathered data to provide a comparison to an established annual BSIMM software security survey, and to a survey about agile practices in the context of technical debt management. The results of these surveys are compared. The details and the conduct of the BSIMM survey were evaluated using analytic evaluation criteria set by Séguin et al. (2012).

The BSIMM framework, the main source of empirical software security engineering data, is analyzed against a literature-based evaluation framework, and certain shortcomings are pointed out. The frequencies reported in BSIMM and the Finnish industry survey are directly compared, and similarities and differences are pointed out. The differences were measured by a statistical evaluation; while the BSIMM does not disclose its survey nor the key techniques of its methods, certain approximations were made. The five-point Likert scale used in the Finnish study was noted to produce strikingly similar results with key indicators, so the observed differences can be considered valid and reliable.

4.5.2 Contribution and future work

The general principles of the BSIMM survey and the resulting maturity model are generally found to concur with the Finnish survey. The methods in the BSIMM's data gathering and the construction of the BSIMM maturity model are subject to criticism. The surveyed impact of the selected security

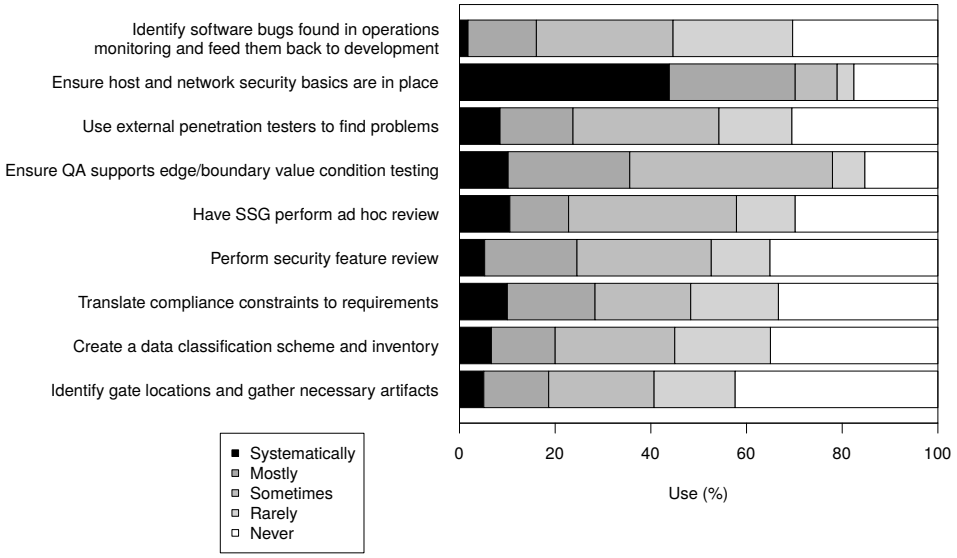


Figure 4.3: Breakdown on the Use of Selected Security Engineering Activities (from P4)

engineering activities is utilized to estimate the benefits received from the 12 core practices. The use and impact of ten of these activities are presented in Figure 4.3 and Figure 4.4.

Respondents were exclusively utilizing agile methodologies in their software development. The results reveal a few characteristics of security engineering in agile software development: first of all, the 'waterfall-like' practice of setting quality gates is rarely used, although found effective. External security testers – penetration testers or the Software Security Group (SSG) – are considered very effective. The external testers may include also QA team in certain organizations. Infrastructure (host and network) security is still considered the most important security practice, although still not considered at all in a significant portion of SW engineering projects.

Based on the comparison it was concluded that certain aspects of the BSIMM do not appear to reflect generic practices in the software industry. It appears evident that the BSIMM has a bias towards a more heavily regulated set of companies, resulting in an atypical set of security assurance requirements and resulting practices, as compared to the software industry in general. An increase in information and software security regulation is an ongoing trend, mandating future empirical research about the ways to efficiently fulfill the regulative security requirements.

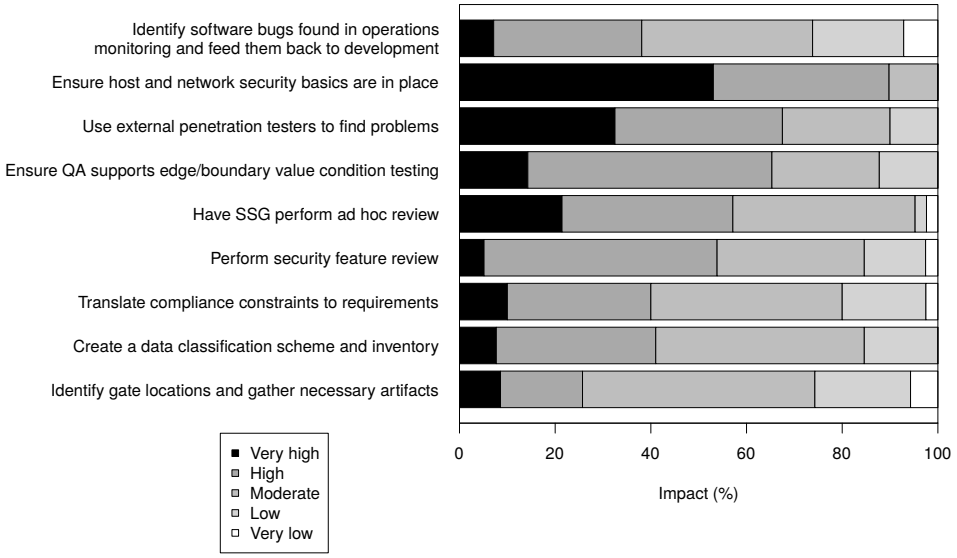


Figure 4.4: Breakdown on the Perceived Impact of Selected Security Engineering Activities (from P4)

4.6 P5: Case Study of Agile Security Engineering: Building Identity Management for a Government Agency

In this study, the need for a practical case example of an agile security project was addressed by an industry project in which Scrum was used to produce a security standard compliant end product. The article describes a case of a large ICT service provider building a secure identity management system for a sizable government agency. The project was subjected to strict regulatory compliance requirements due to the end product's critical role, with an extensive set of specific security requirements.

The target project was a multi-team, multi-site, standard-regulated security engineering and development work executed following the Scrum framework. The study reports the difficulties in combining security engineering with agile development, provides propositions to enhance Scrum for security engineering activities. Also, an evaluation of the effects of the security work on project cost is presented.

4.6.1 Method

This study follows a case study design method by Yin (2003) and a qualitative research approach by Creswell (2003). For the study, we were looking

for a development project that was both using agile methods and fulfilling VAHTI regulations. The case was selected as a representative candidate, able to produce rich information about the phenomenon under study. A case study was conducted along the guidelines set by Wohlin et al. (2012) as a semi-structured interview of the key personnel, and post-hoc observations.

For this study, a post-implementation group interview was held for the key personnel of the selected project. We used a semi-structured interview approach where time was given to the interviewees to elaborate their thoughts about the phenomenon under study. The general questions concerned the scope and size of the project, amount of the personnel involved, and the daily routines of the team. More specifically, the cost of security actions, in work hours, and the proportion of security-related tasks in the sprints were inquired from the project manager responsible for the finances.

4.6.2 Contribution and future work

This study presented a case of building an infrastructure and setting up an identity management software platform for a governmental customer. The customer agency had its own set of security regulations and requirements, the VAHTI instructions. In addition to the government requirements, the ICT service company contracted to build the system was committed to several international ISO/IEC standards, as well as their own management frameworks and sometimes complex financial reporting rules. Both the agency and the service provider's project management offices required employing the Scrum methodology as the project management framework. The research was conducted in a post-project semi-structural interview, and the information was gathered based on their experiences and notes of the project. The parties involved are anonymized, and only publicly available information about the project and the regulations involved was to be disclosed.

Scrum was initially applied in its standard form, with no formal security extensions. Security engineering activities were integrated into the product backlog, and performed within sprints whenever possible. During the project, the team adapted to the security work by creating a *de facto* security developer role, and many of the security engineering tasks ended up to be performed outside of the regular sprint structure: typically, security assurance is based on evidence gained through security testing, which also, in this case, had an adverse effect on the team's ability to schedule and time-box the items that were subject to these tests; these were performed as spikes instead, in which the security personnel performed their tasks individually or in small teams.

The same technique of using spikes was also applied to documentation. Documents were produced outside the main sprints. Security audits and

reviews were also run as separately scheduled one-time tasks. The results of these spikes were nevertheless presented in sprint demos among the other deliverables. The reported issues at product deployment in the production environment prompt for developing and applying a delivery model that provides the required security assurance without the interruption to iterative development. The Scrum process with extensions and outputs required in a security-oriented project is shown in Figure 4.5.

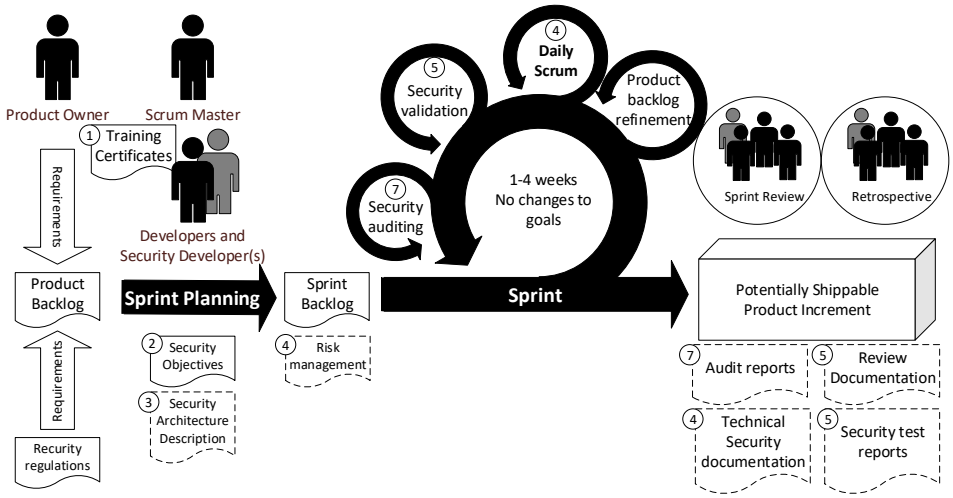


Figure 4.5: Security-oriented Scrum framework and roles (adapted from P5)

The diagram contains the processes and outputs for security assurance (numbered items), as well as the security engineering processes and roles. The observed team viewed the use of Scrum as a positive factor to project cost and quality, although arguably Scrum was not utilized to the maximum extent: important parts of the work were done in spikes outside of the main sprint flow, without attempts to utilize the experience gained from them to time-box the future tasks. This was seen to benefit the project, although an iterative and more exploratory approach to those tasks might have proved more benefits in the long term, and it is still a possibility that the experience gained in this project can be utilized in similar future projects. The project team still regarded the security engineering activities and providing the required security assurance to compose a significant amount of extra work: at final stages, the workload effectively doubled. The initial approach in this project was more or less an unmodified textbook example of the Scrum method, but the team adopted and applied certain security extensions during the project. Additionally, conducting weekly product backlog refinement sessions was deemed essential for the project's success.

The project can be considered to be a model case of two large entities that have decided to fit their traditionally 'non-agile' organizations to work according to an agile framework. The nature of work itself has not changed, although the introduction of a growing amount of security engineering and increasing regulation put an additional strain on the project's requirement management process. Agile methods have an inherent preference to produce working solutions instead of spending time documenting them; in contradiction to this goal, the documentation of the solutions is a key deliverable in the field of security. Scrum will continue to be used by both organizations, and as the team's experience grows, we expect also the cost of the secure systems development to drop, while their quality and security get better.

Based on the experiences gained in this case, Scrum has shown the potential to be suitable for security-oriented development work. With certain additions and modifications, it can be used to provide the security assurance required by the regulators in the ICT and software industry. Especially when applied by an organization capable to adjust itself to fully utilize the flexibility of incremental agile frameworks, instead of partially reverting back to the sequential mode of operations. We are yet to observe a pure agile project where security standards are in a central role: truly integrating security engineering processes and security assurance activities without losing the agile values and benefits gained by the use of those methods is still work in progress.

4.7 P6: Security in Agile Software Development: a Practitioner Survey

To gather further empirical evidence about security in agile software development, an industry survey was conducted among the Finnish software practitioners. The aim of the questionnaire was to answer the three research questions: (1) how is security engineering implemented in software development; (2) what is the perceived impact of software security activities with respect to their level of systematic use; and, (3) do the usage of agile practices and security engineering activities correlate?

Invitations to the survey were sent to over 300 software company CEOs and CIOs, and nearly 1500 security and software practitioners in early 2018. From this group, 62 complete and valid answers were received, 40% of which came directly from developers. The survey revealed that the use of agile software development has an effect on the selection of security engineering practices. The perceived impact of the security practices was lower than the rate of usage would imply, suggesting a selection bias caused by either peculiarities in the security engineering practices, or difficulties in applying the security engineering practices into the iterative software development

workflow.

The conclusion was that security engineering in software development mostly conforms, or is possible to fit in with agile practices. Usage of agile practices correlates with the usage of security engineering: when the use of activities categorized as security engineering practices increases, the use of agile practices is also more systematic. The use of security engineering is concentrated in the implementation phase. Results show a discrepancy between the usage and perceived security impact of the activities, indicating a need for better integration of security engineering methods and tools into the processes and toolchains in software development.

4.7.1 Method

The objective of the study was to empirically verify the usage and impact of software security engineering and accompanying agile software development practices, currently utilized in industry software development projects. This was done by designing an online survey following principles by Pfleeger and Kitchenham (2001), and Kitchenham and Pfleeger (2002).

The survey (N=62) was performed among software and security engineering practitioners, regarding their usage of 40 common security engineering practices in accordance with agile software development activities, processes and artifacts. The survey was conducted following the criteria set by Séguin et al. (2012). The perceived security impact of the used security practices was also surveyed. The results were reported using an analytical framework drawn from extant literature, and research suggestions are provided. Answers were provided on 5-point Likert scale, measuring both the usage of the activities (5=used systematically, 1=not used) and, similarly, the perceived effect of the used activities.

4.7.2 Contribution and future work

This empirical study charted the current use of security activities in agile software development. It also measured the impact and extent of security standards to agile software development processes. The survey revealed two important aspects in software security: about half of the security work done by the software security practitioners is regulation-driven; standards, guidelines and formal requirements drive the software security work. A significant portion of the companies had composed their own security frameworks based on the best practices; only a minority reported that no security regulation was in place.

The study surveyed two aspects in software engineering: first, the use of agile practices, in the form of activities, processes, and artifacts; second, the use of software security engineering activities, practices, and techniques in

conjunction with the software development. Figure 4.6 shows a composition of the agile usage as *degree of agility*.

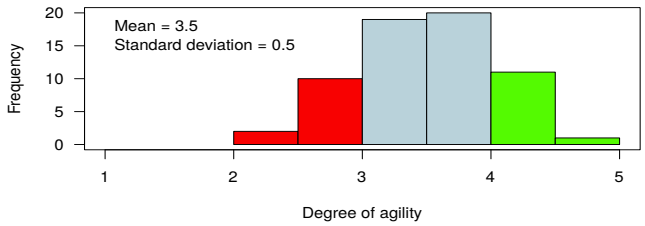


Figure 4.6: Degree of agility (from P6)

This observation of high agile usage, combined with the fact that 0 respondents selected the option "Agile methodologies were not used" when asked for their work method, indicates a broad emerging practice of agile software security engineering in the industry. Agility was further analyzed by examining the correlation between the use of agile practices and individual security activities. This correlation is given in Figure 4.7.

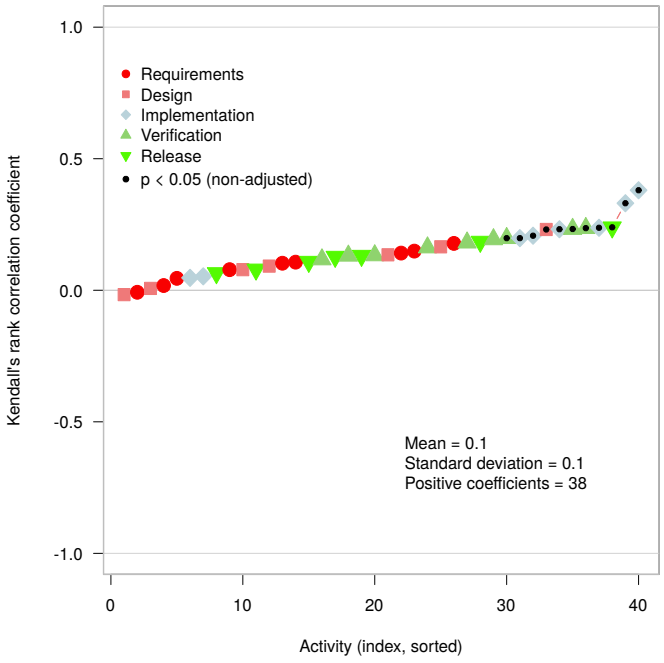


Figure 4.7: Correlations between the degrees of agility and use of security engineering activities

The diagram shows the security engineering practices, most used in conjunction with agile activities, occur on the later stages in the SSDLC: imple-

mentation, verification and release. In contrast, requirement planning and software security design are much less frequent. This may be taken that the architectural and design activities are considered less effective. Research in these fields of software security might reveal possible shortcomings and provide improvement. Figure 4.8 gives a statistical analysis of the observation made already in P5: the correlation between use and the perceived impact of security engineering activities.

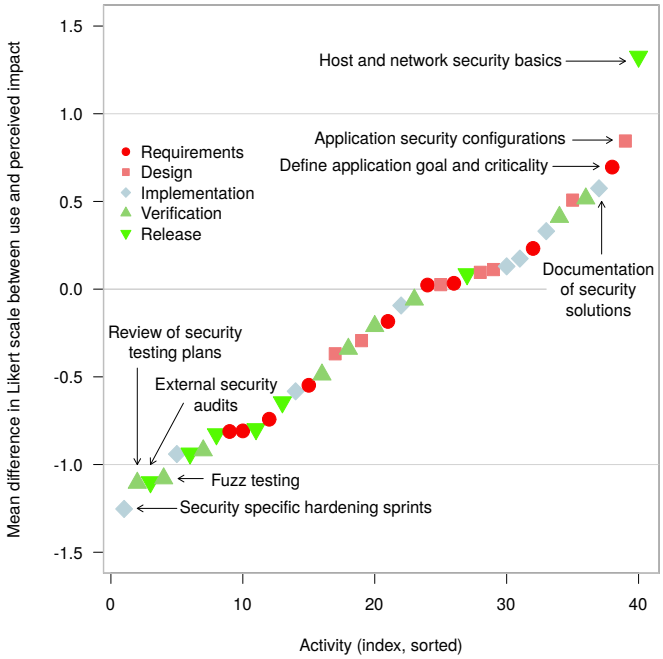


Figure 4.8: Use and perceived impact of security engineering activities (from P6)

A negative correlation indicates a practice that is either underused, or ineffective. In each case, the cause must be individually derived: for example, extra workload and the resulting scheduling pressures are considered likely to inhibit the use of hardening sprints and fuzz testing. On the upper scale, infrastructure security (host and network) was both the most frequently used and considered to be effective. The most used design activity was a generic criticality definition: this activity is naturally the basis of all security work and provides the rationale for security engineering.

The main conclusion is two-fold: first, software development organizations should be more proactive in updating their processes; second, the research community working on software security development models and certificates should pay high attention to their adaptability. Further, a trend in the application of specific security activities to software development can

be observed from the results presented. It was also observed, that security activities executable as part of normal software development activities are more systematically used. Agile practices were found to be extensively used among the respondents. The use of security practices appears to correlate with the increased use of agile practices. However, the extant software security development life cycle models do not appear to guide the security engineering work: practices appear to be selected in conjunction with the software development practices instead. This calls for further research in software security engineering methodologies and techniques.

To facilitate the results, further empirical data needs to be gathered. This includes both quantitative development efficiency data, and static security breach analysis data, as well as qualitative data on the stakeholder's argumentation, relating to how security is defined and attained in their software projects. Direct interviews with the security practitioners, combined with a series of industry surveys targeted to specific groups, will provide an overview of the best practices in the target areas, and the specific needs for further development. The studies should also be conducted in an international scope, to allow comparison.

Chapter 5

Conclusions

The research presented in this thesis focuses on integrating and combining agile software development and security engineering. This involves mixing and matching iterative, incremental and lightweight software development practices with security processes and activities. The security processes and frameworks as well as the software development methodologies are derived from international security standards and industry best practices. The work is based on a combination of peer-reviewed scientific literature, observed industry best practices, and extant professional literature for security experts. The resulting combination of system engineering, security engineering, and software engineering, provides an extensive overview of software security engineering. The provided overview of methodologies enables software development organizations to adopt practices that help producing more secure software and achieving compliance with a wide range of software security standards and regulations.

Security regulation, protecting the data and intellectual property of companies and private citizens, sets the baseline requirements for software security. As a result, software development has additional quality requirements, but security cannot be implemented using traditional quality and safety engineering methods only. Software security engineering provides these methods to both implement the security functionality, and produce the security assurance to prove compliance with the regulation. This research was conducted to examine the security requirements and the means to meet them in a way most suitable for the software developers to adapt their existing development practices and methodologies to security work.

Security engineering, as specified in the security standards, predominantly relies on formalized processes and outcomes. In contrast, since the late 1990s and early 2000s, the focus of the management of software development has shifted into customized and purpose-built processes. Specific focus is in the details of the process, elimination of the overhead, and metic-

ulous management of the work processes. Too generic security requirements for the software products have the effect of creating impediments to the processes without direct security benefits. These formalities are often inherently alien to the new type of software development work, and to the software engineering specialists performing it. Security requirements, increasingly normative and emphasizing assurance, require resources not all organizations possess, thus detrimental to the effectiveness of the software development process. Inappropriate or irrelevant security objectives do not necessarily increase the actual software security, as they may act as a diversion, consuming resources otherwise utilizable for the purpose of meeting more relevant security objectives.

As a result of these observations, the research idea was to apply the agile principles into the software security engineering work: create ways to define minimum viable security, starting from an essential set of requirements, and incrementally adding to it as the implementation progresses. Another important characteristic of current software engineering, largely unacknowledged by the security engineering specifications, is the pervasive use of automated tools. As the research progressed, it became increasingly clear that the original intention of security assurance was to create it automatically, by implementing logging and monitoring systems. While development-time documentation is an important component in both validation of the security implementation and verifying it, development-time verification produced by automated security tools is not a part of any observed security assurance requirements.

The research objective, how to produce secure software, was examined from both theoretical and empirical aspects. Software security standards were used to determine the security objectives and to provide definitions and scope for security assurance. Security standards represent the software industry's best practices in setting security objectives. However, developments in software engineering methodologies and techniques have significantly reduced significance and direct applicability of standardization in the development processes. Principles and paradigms may remain largely immutable, but the implementation processes, techniques, and tools have considerably evolved.

The objective was approached through three research questions:

RQ1: How are software security objectives and practices aligned with agile software development?

RQ2: How can software security be integrated into agile software development methodologies?

RQ3: How is security engineering affecting the agile development process and the security of the software?

Within the research, the individual security objectives and the software security engineering methods, were derived from the regulation and standards. The basic concepts of software security were established. These include security management, security engineering, security assurance engineering, and security risk engineering. Security risk engineering is used to identify the protected assets, and the security threats and vulnerabilities along with their projected impact. Security management involves defining a security rationale through setting security objectives, and enabling the organization to achieve them. Security engineering provides the methods, processes and techniques to achieve the objectives and verify them. Security assurance engineering provides the organization with definite proof of security, resulting from verification and validation of the security engineering work and its end products.

5.1 Results

Selecting the most suitable combination of methodologies is not merely mixing up a variety of security techniques: it involves combining the essential objectives, processes and even philosophies guiding them. Building an efficient process to build secure software is a task of combining the state of the art methods from both software engineering and security engineering traditions. Security assurance for this process is essential.

The standardized security engineering frameworks, as well as the industry maturity models, aim to increase security by defining a broad set of predefined and organization-wide security processes and practices: Security is always accompanied by assurance, and the *compliance* of the output is proved by formality and repeatability of documented processes. This approach also doubles as a risk management technique: in addition to managing the risks by mitigating or avoiding them, they are partially transferred to the certifying party: Security incidents may still happen, but the certificate acts as evidence of the security effort.

A certified process alone does not guarantee secure products: this is achieved by the thoroughness of risk assessment, correctness of the implementation, and appropriate level of verification and validation. Even in the case of formal compliance requirements, *process efficiency* is of utmost importance. Maturity models claim to improve efficiency by promoting repeatability; however, they also burden the organization with processes that may not fit dynamic software development work. The key to effectiveness is not only an efficient process, but an adaptive one that can be adjusted to the changing software requirements. The following sections show what kind of issues and possible solutions were found in this research by answering the research question.

5.1.1 RQ1: Security Objective Alignment to Agile Software Development

Security objectives form the security rationale of the organization and the software product it is creating. The approach to find out how the software security objectives and practices are aligned with agile software development was two-pronged: First, the frameworks and best practices for security engineering were established. The basic set of software security life cycle models and maturity models was established by literature reviews. Second, the security engineering processes, as well as practical examples of using them in agile software development were found from literature descriptions of empirical cases (P1 and P2). This was verified in the empirical part of this research, in publications P5 and P6.

The security objectives were found to be defined and achieved by applying security engineering processes: Context is set by the security regulation, organizational policies, and the utilized technologies. Risk management process is used to explicitly identify and asses the security scope. Security engineering practices are utilized in a security engineering process to create the security controls to mitigate the risk. Appropriate security assurance is produced by the process of verifying and validating the achievement of the security objectives. These were the main subjects of the publications P1 and P2, which directly contribute to this Research Question.

Agile methods are an effective tool in achieving security objectives. While the processes in the security maturity models, such as SSE-CMM and SAMM, are not specifically designed for agile software development, literature-based examples of achieving maturity levels using agile methods prove this possible. In cases where regulatory compliance is a key requirement, the core practices defined in BSIMM may provide an appropriate security engineering framework. However, when the objective is to improve the security of software produced using agile software development methods, the software security development life cycle models provide a set of security practices adjustable and adaptable to suit the organization's needs.

The SSDLC models – Microsoft SDL, and the life cycle models found within the maturity models (SAMM, VAHTI and BSIMM's Touchpoints) – contain the current best practices in software security engineering and security assurance engineering. The risk engineering process was in most models left undefined, despite its importance. Maturity models give a certain level of freedom for implementation, and the actual compliance requirements for achieving higher SSE-CMM maturity levels remain strictly implementation specific and dependent on the evaluation process. No direct contradiction between the security models and agile methods were found; the agile values and principles, however, discourage mechanistic process approaches. Empirical evidence strongly suggests increased security and efficiency achieved

by applying agile principles to the security engineering processes.

5.1.2 RQ2: Integrating Security Engineering into Agile Software Development

The first research question established the methods and context. The majority of the work concentrated on finding out how software security engineering processes can be integrated into agile software development methodologies. This question considers the technical details of the security engineering and software engineering practices concretely applied to the agile software development process. The theme of this question is the efficiency of software security engineering: this approach was selected principally to find out and provide means to integrate the practices, and to ensure the security of the software product: achieving the security objectives. The means are provided from industry practices and extant literature, and justified by the security rationale in the software development context.

The established prevalence of SSDLCs and the key maturity models provide the main source of security engineering practices. The agile software engineering practices were established in a similar process used to answer RQ1: a conceptual work based on the Scrum model (P2, P3), and a comparison of practical survey results (P4). The SSDLC models were used to divide the security engineering practices into life cycle phases. This categorization is more or less artificial when applied to the practical agile software engineering process, but helps to establish their intended purpose. The wideness of the research and the multitude of applied research methods alleviates this by approaching the subject from several different aspects.

Direct counterparts for security engineering practices can be found from the most prevalently used agile practices. The agile practices aim for efficient high-quality products, and were found to provide ample opportunities for introduction of security engineering techniques and practices. The underlying issues primarily concern the mechanisms to prioritize the security objectives and the skills, processes, and tools to achieve them.

5.1.3 RQ3: Challenges and Benefits in Applying Security Engineering to Agile Software Development

This part of the research provided empirical evidence of the challenges and benefits in applying security engineering to agile development. A significant part of the original research motivation was the observed overhead introduced by security requirements to an efficient and working agile process: This was observed in a case study (P5). The security objectives were not necessarily clear or well-communicated, and the security engineering work was performed in isolation by security experts, as an external process sepa-

rate from the actual iterative development cycle. The effect of agile software development methods to software security is considered by analyzing the corresponding techniques and activities.

To find effective industry practices, and to establish the current state of the art in agile security engineering, an industry survey was conducted (P6). In this survey, the software engineering practitioners were explicitly asked about their use of both agile and security engineering practices in concert. The survey was also designed to provide important information about the *impact* of the used security engineering activities. This provides direct input useful in implementing a software security engineering process.

Modern software development is predominantly characterized by the use of agile methods, but security engineering is still mainly driven by compliance requirements. Based on this research, main challenges are lack of security skills among the software engineers, and security engineering's lack of compatibility with agile processes. Key agile principles, such as a focus on high quality and knowledge sharing, directly benefit the security engineering process, and help to achieve the security objectives. While the impact of security engineering practices was generally positive, the impact did not always correlate with the frequency of use. Performing a specific security engineering activity may be detrimental to achieving a technical security objective: Diverting resources to achieve compliance may lead to some security controls, crucial to software security, left unimplemented or inadequately validated and verified.

5.1.4 Summary of results

The standardized security frameworks do contain the essence of security engineering work: to work in software development, security engineering processes must fit in the software engineering ones, not the other way around. Security frameworks define the security objectives, forming the rationale that provides the *justification* for security engineering work; in software security, the proper term for this would be *minimum viable security*. Viable in this context does not mean only compliance or assurance, but concretely achieving the security objectives by accurate and correct software security engineering processes. This concept is considered to be the minimum set of viable security objectives, forming a justifiable security rationale. Implementation is verified and sufficient security assurance is produced.

To achieve the security objectives, a concrete *definition of done* needs to be set. Organizational security processes may be in place, but only to be engaged when necessary; some security engineering tasks may still be executed by security professionals, but security is everybody's responsibility; security-related tasks and requirements are diligently implemented and verified. This is achieved by executing security management and security risk

processes, increasing the level of security awareness by training, and utilizing the achieved security skill in security engineering and security assurance production. By combining the efficient agile and lean software development method processes with the forma approach of security engineering, the best of both worlds can be achieved: provably secure software produced efficiently and sustainably.

Using agile methods for security engineering may require technical and organizational adjustments. In the case observed (P5), work tasks consisting of purely security engineering activities were conducted outside the agile framework and were not time-boxed within the regular iterations. This mode of work may have been beneficial in that particular case, but it worked also as an indicator of the benefits lost when the agile method was not used. The predictability was non-existent, the participants were not communicating effectively with the other members of the team nor the customer, and thus the experience gained from the work largely remained unshared. Agile methods are a particularly powerful tool for organizational development in work demanding highly specialized skills and dynamic decision making.

Findings of the industry survey (P6) indicate a highly polarized group of software security engineers. The respondents were clustered in the high end of both experience and education spectrum. On the other hand, when the software security engineering processes were executed by the same professionals, software security was carefully considered and diligently implemented. Regulation appears to be a main driving force in software security. As an argument against formal security assurance frameworks, not all the security engineering techniques primarily used for security assurance were among the most effective. These include the methods typically introduced by compliance requirements. Also, the amount of security education offered by the employers for the software professionals in Finland was found to be significantly lower compared to security-intensive organizations abroad.

A connection between the use of agile practices and increased utilization of security engineering was found: increased use of agile processes correlates with increased use of software security engineering. From the traditionalist security engineering point of view this may be seen as surprising: from the agile proponent's point of view, it is a natural consequence of meticulous process and quality improvement with an added interest towards individual skill and effective tooling. It was also found, that security engineering practices directly attachable to software development received the most use. This prompts further development and experimentation work among both software and security development methods and tools.

5.2 Limitations

In this research, focusing on the adaptability of security engineering practices into agile software engineering methodologies, the system security engineering approach is necessarily limited. Specifically this concerns the security engineering practices of the software operating environment at the maintenance and disposal phases of the software lifecycle; also, security management and the principles in defining the security objectives are not covered. The research focus was on the methodological level of software security engineering, and in the mutual implications of introducing security elements into software engineering processes. Developing this combination further is a topic of further research of the required tools and techniques.

On the software engineering side, the same limitations apply: software design and architectures are software engineering research areas directly concerned with security. Developing implementation and verification techniques supporting the development of secure software are of critical interest. The individual techniques were investigated and assessed in publication P2, and their impact was among the variables measured in the industry survey (publication P6); they were not at any point thoroughly analyzed for further development or critique. Education and awareness of security issues among the software developers form the very basis of the software security work. The forms of education, such as security certification or even mandatory training, and the effect of security training, were not included in the research.

The limitations noted in the publications note the restricted availability of peer-reviewed source literature, justifying the inclusion of technical papers as direct sources, and prompting an increase in empirical industry reports. Software engineering research could benefit from combining the research effort in software security, safety, and quality due to extensive similarities in the principles and execution of these fields of engineering. On the security side, the effectiveness of security regulation and security assurance remains largely unresearched: quantitative and qualitative data from actual effectiveness and efficacy of the regulated elements is sporadic and not publicly available.

The security techniques were covered by a literary review of agile software security engineering cases (P1), a case study of a Scrum project executed in compliance with security regulations (P5), and an industry survey (P6) about the use and impact of security engineering practices in agile software engineering. Reporting accuracy regarding the actual implementation on these was necessarily limited due to the methodological scope of the source material in P1 and the requirement to comply with the security limitations in P5. The conducted industry survey (P6) also hinted at a limited scope of software security work in the software industry: the respondent profile

was strongly biased towards the most experienced and educated, and consequently most capable people in the industry. Empirical profiling of security work and the effects of security education is required. Similarly, the causality of the relationship between security and agile practices cannot be reliably established without further research.

5.3 Future work

Combining software engineering and security engineering on a methodological level opens several avenues for research, many of them covered in the section describing the limitations. Each development phase in the security development lifecycle provides room for improvement over current software and system engineering practices. This involves iterative and incremental risk management and design practices, as well as security implementation, and verification and validation practices. Continuous delivery models tie the security incident management to the development life cycle and calls for specialized tools with a security aspect, as well as efficient work management and organizing methods. Security assurance should similarly be automated and kept up to date with the actual security controls and structures currently utilized; security assurance can be useful in three roles: security evidence, security guidance, and security forensics – preferably all three at once. Defining security rationale and setting the security objectives is a combination of understanding the security threats and the value of the information assets to be protected. In software security engineering these are to be combined with the resources at disposal as effectively as possible: efficient and systematic security management is a foundation of an organization's software security. Security education creates awareness which guides and motivates the security work.

Much of software engineering work and publications are purely technical and not published on academic forums. Convergence of industry and academia is beneficial for both. Theoretical work and practical innovations follow and enable each other. Merging practices and shared information is beneficial for all: this should include the principles guiding information security and especially security regulation. In software security, quantitative and qualitative data should be the driver for the development of both regulative and technical frameworks. Efficiency and security are tangible, yet elusive goals: effectiveness and competitiveness can be maintained only by constant, ongoing change.

A major part of the future work is practical and empirical research. The main topics in these areas are security architectures, security techniques, and security tools. Metrics, as a component of security improvement, and security verification and security management, provide further interesting

research topics. The security maturity models suggest the security effort be measured, with quite fragmentary practical suggestions to what should be measured, and how. To measure the effectiveness and efficacy of security, metrics, and monitoring are to be developed. Engineering of security requirements and objectives is directly connected to system engineering, and the iterative and incremental models are yet to be applied into this field. Guiding the software development work by security rationale as the minimum viable security could benefit from aspect-oriented and agent-oriented design and implementation techniques. These are all bound together by the use of a development methodology and security activities, which were the focus area of this research.

Developing secure software is a result of skill and resources combined into a managed process. The strategic goal is improved software security using pre-emptive means: risk mitigation where it can be done most effectively. The strategic goal is achieved by education and awareness, and by methodological and technological innovation. The research presented in this thesis provides a framework for implementing this security strategy: Techniques to assess and analyze security risk, to form a security rationale, and to produce provably secure software efficiently.

Software security is in an ongoing competition with constantly evolving security threats. The theory of natural selection by Charles Darwin (1859) appears directly applicable to information security in organizations and software projects: The ones that succeed are the ones most adaptable to change. The evolution of methodologies has defined new prerequisites, into which software security engineering and security regulation will now have to adapt.

Bibliography

Abrahamsson, P., Babar, M. A., and Kruchten, P. (2010). Agility and architecture: Can they coexist? *IEEE Software*, 27(2):16–22.

Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. (2002). Agile software development methods: Review and analysis. *VTT publication 478*.

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

Adelyar, S. H. and Norta, A. (2016). Towards a secure agile software development process. In *2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)*, volume 00, pages 101–106.

Ahmad, M. O., Markkula, J., and Oivo, M. (2013). Kanban in software development: A systematic literature review. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, pages 9–16.

Ambler, S. W. and Lines, M. (2012a). *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press, 1st edition.

Ambler, S. W. and Lines, M. (2012b). *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press.

Anderson, D. J., Concas, G., Lunesu, M. I., Marchesi, M., and Zhang, H. (2012). A comparative study of scrum and kanban approaches on a real case study using simulation. In Wohlin, C., editor, *Agile Processes in Software Engineering and Extreme Programming*, pages 123–137, Berlin, Heidelberg. Springer Berlin Heidelberg.

Anderson, J. C., Rungtusanatham, M., and Schroeder, R. G. (1994). A theory of quality management underlying the deming management method. *Academy of Management Review*, 19(3):472–509.

Anderson, R. J. (2008). *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Publishing, 2nd edition.

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

Ayalew, T., Kidane, T., and Carlsson, B. (2013). *Identification and Evaluation of Security Activities in Agile Projects*, pages 139–153. Springer Berlin Heidelberg, Berlin, Heidelberg.

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

Baskerville, R. and Siponen, M. (2002). An information security meta-policy for emergent organizations. *Logistics Information Management*, 15(5/6):337–346.

Bayuk, J. and Mostashari, A. (2013). Measuring systems security. *Systems Engineering*, 16(1):1–14.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

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*.

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

ben Othmane, L., Angin, P., Weffers, H., and Bhargava, B. (2014). Extending the agile development process to develop acceptably secure software. *IEEE Transactions on Dependable and Secure Computing*, 11(6):497–509.

Beznosov, K. and Kruchten, P. (2004). Towards agile security assurance. In *Proceedings of the 2004 Workshop on New Security Paradigms*, NSPW '04, pages 47–54, New York, NY, USA. ACM.

Blackburn, J. D., Scudder, G. D., and Wassenhove, L. N. V. (1996). Improving speed and productivity of software development: a global survey of software developers. *IEEE Transactions on Software Engineering*, 22(12):875–885.

Boehm, B. (2006). Some future trends and implications for systems and software engineering processes. *Systems Engineering*, 9(1):1–19.

Boehm, B. and Turner, R. (2003a). *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, New York.

Boehm, B. and Turner, R. (2003b). Observations on balancing discipline and agility. In *Proceedings of the Agile Development Conference, 2003. ADC 2003*, pages 32–39.

Boehm, B. and Turner, R. (2004). Balancing agility and discipline: evaluating and integrating agile and plan-driven methods. In *Proceedings. 26th International Conference on Software Engineering*, pages 718–719.

Boehm, B. W. (1991). Software risk management: principles and practices. *IEEE Software*, 8(1):32–41.

Boström, G., Henkel, M., and Wäyrynen, J. (2005). Aspects in the agile toolbox. In Bergmans, L., Gybels, K., Tarr, P., and Ernst, E., editors, *Software Engineering Properties of Languages and Aspect Technologies*.

Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K., and 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, pages 11–18, New York, NY, USA. ACM.

Bruegge, B. and Dutoit, A. H. (2003). *Object-Oriented Software Engineering: Using UML, Patterns and Java, Second Edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Brunet, P. (2000). Kaizen in japan. In *IEE Seminar. Kaizen: From Understanding to Action (Ref. No. 2000/035)*, pages 1/1–1/10.

CCRA (2017a). The common criteria part 1: Introduction and general model, version 3.1.

CCRA (2017b). The common criteria part 2: Security functional components, version 3.1.

CCRA (2017c). The common criteria part 3: Security assurance components, version 3.1.

Chapple, M., Stewart, J. M., and Gibson, D. (2018). *(ISC) 2 CISSP Certified Information Systems Security Professional Official Study Guide*. John Wiley & Sons, 8th edition.

Cleghorn, L. (2013). Network defense methodology: A comparison of defense in depth and defense in breadth. *Journal of Information Security*, 4(3):144–149.

Cockburn, A. (2000). *Writing effective use cases, The crystal collection for software professionals*. Addison-Wesley Professional.

Cockburn, A. (2002). *Agile Software Development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Cockburn, A. (2006). *Agile Software Development: The Cooperative Game (2^{nd} Edition) (Agile Software Development Series)*. Addison-Wesley Professional.

Conboy, K., Fitzgerald, B., and Golden, W. (2005). Agility in information systems development: A three-tiered framework. In Baskerville, R. L., Mathiassen, L., Pries-Heje, J., and DeGross, J. I., editors, *Business Agility and Information Technology Diffusion: IFIP TC8 WG 8.6 International Working Conference May 8–11, 2005, Atlanta, Georgia, U.S.A.*, pages 35–49, Boston, MA. Springer US.

Creswell, J. W. (2003). *Research Design: Qualitative and Quantitative and Mixed Methods Approaches*. SAGE Publications, Inc., Thousand Oaks, California, 2nd edition.

Cruzes, D. S., Felderer, M., Oyetoyan, T. D., Gander, M., and Pekaric, I. (2017). How is security testing done in agile teams? a cross-case analysis of four software teams. In Baumeister, H., Lichter, H., and Riebisch, M., editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 201–216, Cham. Springer International Publishing.

Darwin, C. (1859). *On the origin of species by means of natural selection*. Murray, London.

Davis, N. (2005). Secure software development life cycle processes: A technology scouting report. Technical report, Carnegie-Mellon University Software Engineering Institute, Pittsburgh, PA.

De Win, B., Vanhaute, B., and De Decker, B. (2002). Security through aspect-oriented programming. In De Decker, B., Piessens, F., Smits, J., and Van Herreweghen, E., editors, *Advances in Network and Distributed Systems Security: IFIP TC11 WG11.4 First Annual Working Conference*

on Network Security November 26–27, 2001, Leuven, Belgium, pages 125–138, Boston, MA. Springer US.

Department of Defence Information Analysis Center (2018). DoD Cybersecurity Policy Chart, December 2018.

Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM*, 15(10):859–866.

DoD (1985a). *GUIDANCE FOR APPLYING THE DEPARTMENT OF DEFENSE TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA IN SPECIFIC ENVIRONMENTS*. United States Department of Defence.

DoD (1985b). *TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA*. United States Department of Defence.

Dorca, V., Munteanu, R., Popescu, S., Chioreanu, A., and Peleskei, C. (2016). Agile approach with kanban in information security risk management. In *2016 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, pages 1–6.

Elahi, G., Yu, E., Li, T., and Liu, L. (2011). Security requirements engineering in the wild: A survey of common practices. In *2011 IEEE 35th Annual Computer Software and Applications Conference*, pages 314–319.

Felderer, M. and Schieferdecker, I. (2014). A taxonomy of risk-based testing. *International Journal on Software Tools for Technology Transfer*, 16(5):559–568.

Fitzgerald, B., Stol, K., O'Sullivan, R., and O'Brien, D. (2013). Scaling agile methods to regulated environments: An industry case study. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 863–872.

Fitzgerald, B., Stol, K.-J., O'Sullivan, R., and 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, pages 863–872.

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

Glass, R. L. (2003). *Facts and Fallacies of Software Engineering*. Addison-Wesley Professional.

Hamidovic, H. (2012). Fundamental concepts of IT security assurance. *ISACA Journal*, 2:45.

Herath, T. and Rao, H. (2009). Encouraging information security behaviors in organizations: Role of penalties, pressures and perceived effectiveness. *Decision Support Systems*, 47(2):154 – 165.

Hevner, A. and March, S. T. (2004). Design science research in information systems. *MIS quarterly*, 28(1):75–105.

Holvitie, J., Licorish, S. A., Spínola, R. O., Hyrynsalmi, S., MacDonell, S. G., Mendes, T. S., Buchan, J., and Leppänen, V. (2017). Technical debt and agile software development practices and processes: An industry practitioner survey. *Information and Software Technology*.

Howard, M. (2004). Building more secure software with improved development processes. *IEEE Security Privacy*, 2(6):63–65.

Howard, M. and Lipner, S. (2006). *The Security Development Lifecycle*. Microsoft Press, Redmond, WA, USA.

Huo, M., Verner, J., Liming, Z., and Babar, M. (2004). Software quality and agile methods. In *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, pages 520–525.

Hutchins, E. M., Cloppert, M. J., and Amin, R. M. (2011). Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80–106.

ICS-CERT (2016). *Recommended Practice: Improving Industrial Control System Cybersecurity with Defense-in-Depth Strategies*. U.S. Homeland Security.

Ingalsbe, J. A., Kunimatsu, L., Baeten, T., and Mead, N. R. (2008). Threat modeling: Diving into the deep end. *IEEE Software*, 25(1):28–34.

ISO/IEC Sstandard 27034-1:2011 (2011). *Information technology — Security techniques — Application security — Part 1: Overview and concepts*. ISO/IEC.

ISO/IEC Standard 15026-1:2013 (2013). *Systems and software engineering – Systems and software assurance – Part 1: Concepts and Vocabulary*. ISO/IEC.

ISO/IEC Standard 15026-2:2011 (2011). *Systems and software engineering – Systems and software assurance – Part 2: Assurance Case*. ISO/IEC.

ISO/IEC Standard 15288 (2015). *Systems and software engineering – System life cycle processes*. ISO/IEC.

ISO/IEC Standard 15408-1:2009 (2014). *Information technology - Security techniques - Evaluation criteria for IT security*. ISO/IEC, 3rd edition.

ISO/IEC Standard 15443-1:2012 (2012). *Information technology – Security techniques – Security assurance framework – Part 1: Introduction and concepts*. ISO/IEC.

ISO/IEC Standard 21827:2008 (2008). *Information Technology – Security Techniques – Systems Security Engineering – Capability Maturity Model (SSE-CMM)*. ISO/IEC, 2nd edition.

ISO/IEC Standard 27005:2018 (2018). *Information technology — Security techniques — Information security risk management*. ISO/IEC.

ISO/IEC standard 33001:2015 (2015). *Information technology – Process assessment – Concepts and terminology*. ISO/IEC.

ISO/IEC/IEEE Standard for Systems and software engineering (2010). International Standard - Systems and software engineering – Vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, pages 1–418.

ISO/IEC/IEEE Standard for Systems and Software Engineering Life Cycle Management (2018). International Standard - Systems and software engineering - Life cycle management - Part 1: Guidelines for life cycle management. *ISO/IEC/IEEE 24748-1:2018*, pages 1–82.

Jakobsen, C. R. and Sutherland, J. (2009). Scrum and CMMI Going from Good to Great. In *2009 Agile Conference*, pages 333–337.

Järvinen, P. (2004). Research questions guiding selection of an appropriate research method. Series of Publications D – Net Publications D–2004–5, Department of Computer Sciences, University of Tampere, Tampere, Finland.

Karlstrom, D. and Runeson, P. (2005). Combining agile methods with stage-gate project management. *IEEE Software*, 22(3):43–49.

Kim, G., Debois, P., Willis, J., and Humble, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., and Linkman, S. (2009). Systematic literature reviews in software engineering â€“ a systematic literature review. *Information and Software Technology*,

51(1):7 – 15. Special Section - Most Cited Articles in 2002 and Regular Research Papers.

Kitchenham, B. A. and Pfleeger, S. L. (2002). Principles of survey research part 2: Designing a survey. *SIGSOFT Softw. Eng. Notes*, 27(1):18–20.

Knaster, R. and Leffingwell, D. (2018). *SAFe 4.5 Distilled: Applying the Scaled Agile Framework for Lean Software and Systems Engineering.* Addison-Wesley Professional, 2nd edition.

Kniberg, H. and Ivarsson, A. (2012). Scaling Agile @ Spotify. *https://creativeheldstab.com/wp-content/uploads/2014/09/scaling-agile-spotify-11.pdf.*

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, pages 805–808, New York, NY, USA. ACM.

Kruchten, P. (2000). *The Rational Unified Process: An Introduction, Second Edition.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition.

Kruchten, P. (2010). Software architecture and agile software development: A clash of two cultures? In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 497–498, New York, NY, USA. ACM.

Krutz, R. L. and Vines, R. D. (2010). *Cloud Security: A Comprehensive Guide to Secure Cloud Computing.* Wiley Publishing.

Laukkarinen, T., Kuusinen, K., and Mikkonen, T. (2018). Regulated software meets devops. *Information and Software Technology*, 97:176 – 178.

Licorish, S. A., Holvitie, J., Hyrynsalmi, S., Leppänen, V., Spínola, R. O., Mendes, T. S., MacDonell, S. G., and Buchan, J. (2016). Adoption and suitability of software development methods and practices. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, pages 369–372.

Marcal, A. S. C., Soares, F. S. F., and Belchior, A. D. (2007). Mapping cmmi project management process areas to scrum practices. In *31st IEEE Software Engineering Workshop (SEW 2007)*, pages 13–22.

McGraw, G. (2006). *Software Security: Building Security In.* Addison-Wesley Professional.

Mead, N. (2015). *Security Quality Requirements Engineering (SQUARE).* Software Engineering Institute (SEI).

Mead, N. R. and Stehney, T. (2005). Security quality requirements engineering (square) methodology. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7.

Microsoft (2017). Agile development using microsoft security development lifecycle.

MITRE CVE (2018). National Vulnerability Database.

Mohan, V. and Othmane, L. B. (2016). SecDevOps: Is it a marketing buzzword? - mapping research on security in DevOps. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 542–547.

Moyon, F., Beckers, K., Klepper, S., Lachberger, P., and Bruegge, B. (2018). Towards continuous security compliance in agile software development at scale. In *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, pages 31–34.

NIST (2006). Standard for minimum security requirements for federal information and information systems. Federal Information Processing Standards (FIPS) publication 200.

NIST NVD (2018). National Vulnerability Database.

OWASP SAMM (2017). Software assurance maturity model.

Oyetoyan, T. D., Cruzes, D. S., and Jaatun, M. G. (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)*, pages 548–555.

Patiño, S., Solís, E. F., Yoo, S. G., and Arroyo, R. (2018). Ict risk management methodology proposal for governmental entities based on iso/iec 27005. In *2018 International Conference on eDemocracy eGovernment (ICEDEG)*, pages 75–82.

Pfleeger, S. L. and Kitchenham, B. A. (2001). Principles of survey research: Part 1: Turning lemons into lemonade. *SIGSOFT Softw. Eng. Notes*, 26(6):16–18.

Ponemon Institute (2017). Cost of cyber crime study. *Accenture and Ponemon Institute LLC.*

Ponemon Institute (2018). Cost of data breach study. *IBM Security and Ponemon Institute LLC.*

Poppendieck, M. and Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit: An Agile Toolkit.* Addison-Wesley.

Poth, A., Sasabe, S., Mas, A., and Mesquida, A. (2018). Lean and agile software process improvement in traditional and agile environments. *Journal of Software: Evolution and Process*, 0(0).

Ramesh, B., Cao, L., and Baskerville, R. (2010). Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480.

Rantala, O.-P. and Kievari, T. (2016). Maailman luotetuinta digitaalista liiketoimintaa. suomen tietoturvallisuusstrategia.

Rindell, K. and Holvitie, J. (2019). Security Risk Assessment and Management as Technical Debt. In *Proceedings of the IEEE Cyber Science 2019 Conference, June 3–4, 2019, Oxford, UK*. IEEE.

Rindell, K., Holvitie, J., Ruohonen, J., Hyrynsalmi, S., and Leppänen, V. (2019a). Industry Survey of Secure Engineering Practices in Software Engineering. *Article in Review*.

Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2015a). A comparison of security assurance support of agile software development methods. In *Proceedings of the 16th International Conference on Computer Systems and Technologies*, CompSysTech '15, pages 61–68, New York, NY, USA. ACM.

Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2015b). Securing Scrum for VAHTI. In Nummenmaa, J., Sievi-Korte, O., and Mäkinen, E., editors, *Proceedings of 14th Symposium on Programming Languages and Software Tools*, pages 236–250, Tampere, Finland. University of Tampere.

Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2016). Case Study of Agile Security Engineering: Building Identity Management for a Government Agency. In *Proceedings of Availability, Reliability and Security (ARES), 2016 11th International Conference*, pages 556–563.

Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2017a). Busting a Myth: Review of Agile Security Engineering Methods. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, pages 74:1–74:10, New York, NY, USA. ACM.

Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2017b). Case Study of Agile Security Engineering: Building Identity Management for a Government Agency. *International Journal of Secure Software Engineering*, 8(8):43–57.

Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2018a). Aligning Security Objectives With Agile Software Development. In *In proceedings of XP '18 Companion, May 21–25, 2018, Porto, Portugal*, XP'18, pages 0–0, New York, NY, USA. ACM.

Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2018b). Fitting Security into Agile Software Development. *International Journal of Systems and Software Security and Protection (IJSSSP)*, 9(1):47–70.

Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2019b). Challenges in Agile Security Engineering: A Case Study. In *Exploring Security in Software Architecture and Design*, chapter 12, pages 287–312. IGI Global.

Rindell, K., Hyrynsalmi, S., and Leppänen, V. (2019c). Security Assurance in Agile Software Development Methods: An Analysis of Scrum, XP and Kanban. In *Exploring Security in Software Architecture and Design*, chapter 3, pages 47–68. IGI Global.

Rindell, K., Ruohonen, J., and Hyrynsalmi, S. (2018c). Surveying Secure Software Development Practices in Finland. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, pages 6:1–6:7, New York, NY, USA. ACM.

Rodríguez, P., Markkula, J., Oivo, M., and Turula, K. (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*, pages 139–148.

Ross, R. S., McEvilley, M., and Oren, J. C. (2014). Systems security engineering: An integrated approach to building trustworthy resilient systems. In *National Institute of Standard and Technology (NIST) Special Publication (SP) 800-160*. National Institute of Standard and Technology (NIST).

Schneider, F. and Berenbach, B. (2013). A literature survey on international standards for systems requirements engineering. *Procedia Computer Science*, 16:796 – 805. 2013 Conference on Systems Engineering Research.

Schwaber, K. (1995). Scrum development process. oopsla'95 workshop on business object design and implementation. *Austin, USA*.

Schwaber, K. and Beedle, M. (2002). *Agile Software Development with Scrum*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Schweigert, T., Vohwinkel, D., Korsaa, M., Nevalainen, R., and Biro, M. (2014). Agile maturity model: analysing agile maturity characteristics

from the SPICE perspective. *Journal of Software: Evolution and Process*, 26(5):513–520.

Séguin, N., Tremblay, G., and Bagane, H. (2012). Agile principles as software engineering principles: An analysis. In Wohlin, C., editor, *Agile Processes in Software Engineering and Extreme Programming*, pages 1–15, Berlin, Heidelberg. Springer Berlin Heidelberg.

Shewhart, W. A. and Deming, W. E. (1939). *Statistical method from the viewpoint of quality control*. Courier Corporation.

Silva, F. S., Soares, F. S. F., Peres, A. L., de Azevedo, I. M., Vasconcelos, A. P. L., Kamei, F. K., and de Lemos Meira, S. R. (2015). Using cmmi together with agile software development: A systematic review. *Information and Software Technology*, 58:20 – 43.

Sindre, G. and Opdahl, A. L. (2005). Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44.

Sommerville, I. (2015). *Software Engineering*. Pearson, 10th edition.

Stavru, S. (2014). A critical examination of recent industrial surveys on agile method usage. *Journal of Systems and Software*, 94:87 – 97.

Stirbu, V. and Mikkonen, T. (2018). Towards agile yet regulatory-compliant development of medical software. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 337–340.

Such, J. M., Gouglidis, A., Knowles, W., Misra, G., and Rashid, A. (2016). Information assurance techniques: Perceived cost effectiveness. *Computers & Security*, 60:117 – 133.

Sugimori, Y., Kusunoki, K., Cho, F., and Uchikawa, S. (1977). Toyota Production System and Kanban System Materialization of Just-In-Time and Respect-For-Human System. *International Journal of Production Research*, 15(6):553–564.

Synopsys Software Integrity Group (2018). The building security in maturity model 9.

Takeuchi, H. and Nonaka, I. (1986). The new new product development game. *Harvard Business Review*, pages 137–146.

Tøndel, I. A., Jaatun, M. G., and Meland, P. H. (2008). Security requirements for the rest of us: A survey. *IEEE Software*, 25(1):20–27.

Tsipenyuk, K., Chess, B., and McGraw, G. (2005). Seven pernicious kingdoms: a taxonomy of software security errors. *IEEE Security Privacy*, 3(6):81–84.

Tsipenyuk, K., Chess, B., and McGraw, G. (2005). Seven pernicious kingdoms: a taxonomy of software security errors. *IEEE Security Privacy*, 3(6):81–84.

Türpe, S. and 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)*, pages 34–49. CEUR Workshop Proceedings.

VAHTI (2016). Vahti-ohje.

VAHTI 1/2013 (2013). Sovelluskehityksen tietoturvaohje. Referenced 8th Oct. 2017.

VersionOne (2018). 12th annual state of agile survey. https://versionone.com/pdf/VersionOne-12th-Annual-State-of-Agile-Report.pdf.

Viega, J. and McGraw, G. (2002). *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley, 1st edition.

Wake, B. (2003). Invest in good stories, and smart tasks.

Wäyrynen, J., Bodén, M., and Boström, G. (2004). *Security Engineering and eXtreme Programming: An Impossible Marriage?*, pages 117–128. Springer Berlin Heidelberg, Berlin, Heidelberg.

Win, B. D., Scandariato, R., Buyens, K., Grégoire, J., and Joosen, W. (2009). On the secure software development process: Clasp, sdl and touchpoints compared. *Information and Software Technology*, 51(7):1152 – 1171. Special Section: Software Engineering for Secure Systems.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

Yin, R. K. (2003). *Case Study Research: Design and Methods*. SAGE Publications, Inc., 3rd edition.

Yost, J. R. (2007). 20 - a history of computer security standards. In Leeuw, K. D. and Bergstra, J., editors, *The History of Information Security*, pages 595 – 621. Elsevier Science B.V., Amsterdam.

Yost, J. R. (2015). The origin and early history of the computer security software products industry. *IEEE Annals of the History of Computing*, 37(2):46–58.

I

Paper I

Busting a Myth: Review of Agile Security Engineering Methods

Rindell, Kalle and Hyrynsalmi, Sami and Leppänen, Ville (2017). In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, pages 74:1–74:10.

Busting a Myth: Review of Agile Security Engineering Methods

Kalle Rindell
University of Turku
Turku, Finland FI-20014
kalle.rindell@utu.fi

Sami Hyrynsalmi
Tampere University of Technology
Pori, Finland FI-28100
sami.hyrynsalmi@tut.fi

Ville Leppänen
University of Turku
Turku, Finland FI-20014
ville.leppanen@utu.fi

ABSTRACT

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

KEYWORDS

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

ACM Reference format:
Kalle Rindell, Sami Hyrynsalmi, and Ville Leppänen. 2017. Busting a Myth: Review of Agile Security Engineering Methods. In *Proceedings of ARES '17, Reggio Calabria, Italy, August 29-September 01, 2017,* 10 pages.
DOI: 10.1145/3098954.3103170

1 INTRODUCTION

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

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

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

2 BACKGROUND

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

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

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

The studies surveyed in this article also reference widely to Microsoft Security Development Lifecycle, OWASP CLASP, Cigital Touchpoints and ISO Common Criteria. These security processes and models provide the basic security tools and activities, ready to be adapted and integrated into the software development methods. The models have roots in security engineering practices that

ARES '17, Reggio Calabria, Italy

DOI: 10.1145/3098954.3103170

Table 1: Searches

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

predate the agile boom, leading to initial difficulties with adaption to agile methods. A starting hypothesis for several works, cf. for example [19], is that an agile method in itself is somehow perceived to be incompatible with secure software development. Despite the difficulties, several examples of agile software development with integrated security engineering activities and processes have been presented [e.g. 3, 19, 24]. However, the number of empirical studies assessing and describing industrial cases is low. For example, Rindell et al. [17] studied a case where Scrum was applied in the development of a security-critical system. While they found shortages in converting the security engineering processes into a truly agile process, the project itself was deemed a success: the end product was delivered on budget and on time, while meeting the requirements of the security regulations that applied to the case.

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

3 RESEARCH PROCESS

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

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

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

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

After the library search, a manual screening and selection process was performed, to ensure the validity of the first search round:

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

The articles selected by this two-stage process were then reviewed and analysed, and the security activities described in the articles were identified. These activities were evaluated using the principles of Common Criteria, which is the ISO standard to "permit comparability between the results of independent security evaluations". It does so by providing a common set of requirements for the security functions of IT products and systems and for assurance measures applied to them during a security evaluation" [10]. Furthermore, each method was screened for any empirical evidence provided, as were references to any security, quality or safety standard.

Citation count was hypothesized to project industry or scientific significance. However, the citation count of individual articles outside their respective research groups and direct affiliates was found to be very low with the notable exception of [7]. One of the papers [20], without any citations and certain noted shortcomings, got selected for its merit in summarizing CLASP security process from an agile point of view. The selection of 11 studies is presented and summarized in Table 2.

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

The activities found in each study were placed into a life cycle model, consisting of six distinct phases: Pre-requirement, Requirement, Design, Implementation, Testing and Release. This generic life cycle phase division is used by two of the selected articles [see 2, 4] and resembles closely the 7-stage life cycle model of the SDL:

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

Table 2: Studies selected for review

Topic summary	Standards	Empirical	Cited	Source
Security Engineering and XP	Yes	No	3	Wäyrynen et al. 2004 [24]
Towards Agile Security in Web Applications	No	No	5	Kongsli et al. 2006 [11]
XP Practices and Security Requirements Engineering	No	Yes	10	Boström et al. 2006 [5]
Agile Security Using an Incremental Security Architecture	No	No	4	Chivers et al. 2005 [6]
XP Security Practices	No	No	3	Ge et al. 2007 [9]
CLASP, SDL and Touchpoints compared	No	No	40	De Win et al. 2009 [7]
FISA-XP: Integration of security activities with XP	No	No	0	Sonia et al. 2014 [20]
Agile security management	No	No	6	Baca et al. 2011 [4]
Security Activities in Agile Projects	Yes	Yes	1	Ayalew et al. 2013 [2]
Extending agile methods with security	No	Yes	8	Othmane et al. 2014 [3]
Software Security Skills, Usage and Training Needs in Agile	Yes	Yes	1	Oyetoyan et al. 2016 [16]

the SDL model has an added maintenance phase after the first six. In [7] the phase division is a even more elaborate, with nine stages. The model presented in that study is a combination of several security processes: in addition to an added support stage, the pre-requirement and design stages are each divided in two. However, the six-stage life cycle was deemed to resemble agile software development process with sufficient accuracy and clarity, and was therefore selected.

4 REVIEWS

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

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

4.1 Security Engineering and eXtreme Programming: An Impossible Marriage? [24]

The first and oldest of the selected articles by Wäyrynen, Bodén, and Boström [24] concentrates on extreme programming. It provides an analysis of the XP method from the viewpoint of two central ISO standards for software security: Systems Security Engineering – Capability Maturity Model (SSE-CMM, ISO 21817) and the Common Criteria (CC, ISO 15408). The article provides a solid analysis of the agile security: it discusses the topic of insecurity of agile methods, reviews the literature and earlier work aiming to integrate security activities into agile methods and transform them into agile activities, and then provides an analysis standing on two cornerstones of the traditional security engineering standards.

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

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

The authors acknowledge the need for empiric validation missing from this article. They present a claim that their approach is limited to particular fields of software engineering, specifically excluding real-time software and safety critical applications. The paper is the earliest one of the selected studies, pointing out that certain agile practises are also beneficial security activities. The identified activities were security education, misuse cases, simplicity of (security) design, pair programming, and security testing. The security activities the authors acknowledged to be missing from XP were excluded from the results.

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

The paper by Boström, Wäyrynen, Bodén, Beznosov, and Kruchten [5] enhances XP by introducing two security specific mechanisms: abuser stories as threat scenarios, and security-related user stories as security functionalities. The authors suggest that incorporating the security engineering activities into XP as agile processes,

using XP terminology and mechanisms, yields the intended benefits without sacrificing the agile method's claimed benefits. The security-augmented XP process is explained starting from requirements planning, and introducing a way to integrate the abuser stories and security-related user stories into XP's planning game. Since stories are the essential way of communicating requirements and features in XP, the two new story types affect the whole development process.

The method is applied in a student project, which is described from the security point of view: specifically, the implications of the added features on the XP process are discussed. For the project, they have also introduced the role of a security engineer, which is typically an explicit requirement in various security standards. The Common Criteria is mentioned several times, and its requirements discussed, yet the method or the described student project do not claim compliance with any security standard, nor specifically provide the security assurance that would satisfy the CC requirements. The proposed security activities consist of two key design phase processes: use of misuse cases (abuser stories, security-related user stories), which were considered to have effect when applying security principles to design.

4.3 Towards Agile Security in Web Applications [11]

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

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

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

4.4 Agile Security Using an Incremental Security Architecture [6]

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

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

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

design (in response to changing architecture-level requirements), and performing a security analysis of the system design.

4.5 Extreme Programming Security Practices [9]

Ge, Paige, Polack, and Brooke [9] propose security training and fundamental security architecture for XP as means to establish a security criteria. Security training would effect the team's way of conducting the XP's planning game: team's awareness of security issues would bring security issues into the center of the stage. Specifically, they would write security stories, understand security threats and vulnerabilities, be able to write better code and avoid common mistakes, and be able to perform security testing. Fundamental security architecture, in contrast to an iterative security architecture, is defined before the iterations commence, and outlines the available security mechanisms, system platforms and provides basis for risk and vulnerability assessments. An example representation of a fundamental security architecture would be a collection of engineering patterns. The architecture could be created based on architectural risk analyses, security and programming best practises, and from both tacit and documented software project experience and knowledge.

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

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

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

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

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

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

4.7 Agile Development with Security Engineering Activities [4]

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

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

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

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

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

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

Ayalew, Kidane, and Carlsson [2] have selected several security engineering activities from industry security frameworks: CLASP, SDL, Cigital Touchpoints and the Common Criteria, and performed a survey-based cost-benefit analysis of these security activities. The approach extends the previous work by producing a list of 'agile compatible and beneficial security activities'. This resulting list of 16 activities appears to cover all phases of the secure software development life cycle. Countermeasure graphs, introduced by Baca and Carlsson in [4] are present also in this study, which, coupled with the same the life cycle phase model, is another concrete evidence of the connection between these studies. The similarity between this and the previous studies is noted and discussed, and the extended results and marked differences of this paper are pointed out.

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

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

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

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

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

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

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

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

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

The article presents a methodology which not merely integrates security activities into an agile method, but also offers a method to provide continuous security assurance. Although somewhat limited and without concrete empiric evidence, each incremental release candidate produced by applying the proposed is secure and ready for acceptance. Although complying with security standards is not specifically mentioned in the text, this approach includes all the necessary components for standard compliance. A real world

approach might not aim for continuous security assurance, i.e., transition phase tasks done during each iteration, which could improve the approach by reducing the workload and therefore the cost of security tasks. On the other hand, continuous security assurance makes the proposed method directly eligible for DevOps and other continuous delivery and continuous integration models, characterized by frequent delivery of new software increments into production environment.

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

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

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

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

5 RESULTS

The purpose of this study was to outline and identify the security activities used in agile software development. The papers selected

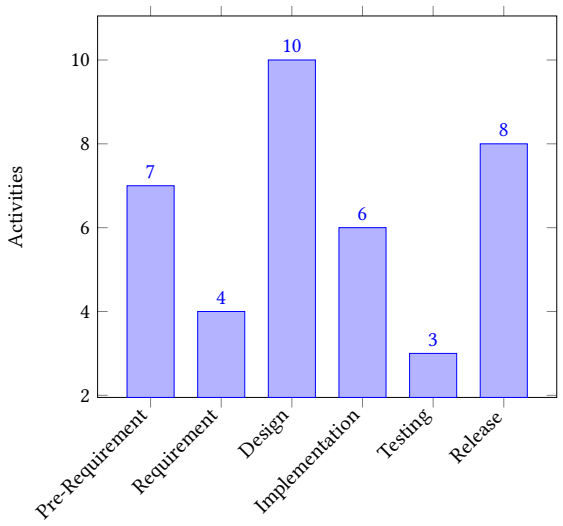


Figure 1: Distribution of the activities by life cycle phase

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

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

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

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

Table 3: Security activities in life cycle phases

Phase	Activity	Occurrences	Source(s)
Pre-requirement			
	Specify operational environment	1	[9]
	Identify global security policy	1	[9]
	Institute security awareness program	3	[9], [24], [2]
	Monitor security practises	1	[9]
	Research and assess security solutions	1	[9]
	Build information labeling scheme	1	[9]
	Project security officer	1	[16]
	Pre-Requirement phase adaptations total	9	
Requirement			
	Identify user roles and requirements	5	[4], [6], [9], [16], [2]
	Perform security analysis of requirements	5	[4], [6], [9], [16], [2]
	Specify resource-based security properties	2	[6], [16]
	Requirement inspection	1	[4]
	Requirement phase adaptations total	13	
Design			
	Risk estimation	3	[3], [16], [2]
	Threat modeling	2	[3], [16]
	Document security design assumptions	5	[4], [6], [3], [16], [2]
	Detail misuse cases	7	[11], [4], [5], [6], [24], [3], [16]
	Apply security principles to design	4	[5], [6], [24], [2]
	Specify DB security configuration	0	
	Perform security analysis of system design	3	[4], [6], [2]
	Design UI for security functionality	0	
	Annotate class designs with security properties	1	[16]
	Quality gates	1	[2]
	Design phase adaptations total	26	
Implementation			
	Coding standards	3	[4], [16], [2]
	Pair programming	2	[24], [16]
	Integrate security analysis into build process	1	[2]
	Implement and elaborate resource policies	0	
	Implement interface contracts	0	
	Address reported security issues	1	[16]
	Implementation phase adaptations total	7	
Testing			
	Identify and implement security tests	6	[11], [4], [24], [3], [2]
	Perform security function usability testing	1	[4]
	Perform SW security fault injection testing	1	[2]
	Testing phase adaptations total	8	
Release			
	Manage system security authorization agreement	1	[3]
	Perform source level security reviews	3	[4], [3], [16]
	Verify security attributes of resources	1	[3]
	Manage certification process	1	[3]
	Perform code signing	2	[3], [2]
	Build operational security guide	2	[3], [2]
	Manage security issue disclosure process	3	[11], [3], [16]
	Repository improvement	1	[4]
	Release phase adaptations total	14	
	Adjusted total number of security activities	38	

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

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

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

6 DISCUSSION

The findings implicate that while adoption of agile software development methods has been perhaps slower in the security field, but nevertheless it has largely already happened. The perceived discrepancy between agile values and security requirements has largely been solved, and both security activities and agile methods – and the experts utilizing them – have adapted to their integrated use. The original intention was not to inspect the agile security 'myth', yet the theme occurred in so many papers, especially in the earlier ones, that we decided the time has come to officially declare the death of of the incompatibility myth. One probable and much-cited origin or at least a propagator of the unsuitability myth is the of Viega and McGraw's book published in 2002 [23], in which a thorough and meticulous pre-planning is emphasized, and agile methods found to be lacking in this aspect. In the light of current studies, however, the conflict appears to be more theoretical than practical or technical.

Certain limitations are acknowledged in the selected approach: the articles were selected from the result set in common digital libraries, and the source material is far from complete. Addition of non-academic sources might have produced rather interesting results and potentially well-documented cases, yet the search of this study was limited to peer-reviewed articles. It is to be noted that an unexplored body of work with further – or contrary – evidence may not have been included in the result set, excluded from the source material. The invariably positive view of agile secure activities in all of the surveyed articles can also be a reflection of publication bias: cases in which the agile methods are found not to be suitable for the task at hand are perceived to be failures and do not get published. Also, access to restricted environments and projects may be limited, and publication of the results withheld. Furthermore, certain types of the software products and organizations may provide easier access to empiric evidence.

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

7 CONCLUSIONS AND FUTURE WORK

In the light of this study, the practise of security engineering appears well adapted to and widely used with agile software development methods. Some of the activities have been modified to better suit iterative development model, and, based on the literature, much attention has been paid to retain the agile nature of the development process despite of the added security activities. The often-repeated myth of agile methods' incompatibility or inherent unsuitability for security tasks has been effectively been broken already in the first observed study, published in 2004 [24] - yet the myth is perpetuated for years after this well up until mid-2010's. The field of secure agile software development is still fragmented and organization specific, largely due to highly adaptable nature of the agile methods. A wide industry survey concentrating on agile security activities would help identifying the key agile security practises and confirm the findings outlined in this study. Also using other than peer-reviewed academic sources would provide an interesting ground for hypotheses, to be verified with a scientific method. Finally, a distinctive similarity between security and safety activities implies an increase in the general quality of the software products created with security-augmented processes. The impact of security activities on the measurable quality of software and software projects should also prove to be an interesting research subject.

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

II

Paper II

Aligning Security Objectives With Agile Software Development

Rindell, Kalle and Hyrynsalmi, Sami and Leppänen, Ville (2018). In *In proceedings of XP '18 Companion, May 21–25, 2018, Porto, Portugal*, pages 0–0.

Aligning Security Objectives With Agile Software Development

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

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

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

ABSTRACT

Success of a software development process is determined by its ability to transform its objectives into requirements, and the requirements features and functionality. In addition to business objectives, software development also has security objectives, requiring security engineering activities. Software security engineering is characterized by sequential life cycle models, in sharp contrast to the iterative and incremental software development processes. This leads to a situation, where security and business objectives are to be met through conflicting approaches. In this study, security engineering activities from Microsoft SDL, the ISO Common Criteria and OWASP SAMM security development lifecycle models are mapped into common agile processes, practises and artifacts, with the intention of decreasing the incompatibility gap between the approaches. The organizational and technical aspects of the mapping are considered primarily from the point of view of achieving the security objectives: setting security requirements for design, implementation and verification, and releasing secure software through efficient software security development processes.

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 *XP '18 Companion: XP '18 Companion, May 21–25, 2018, Porto, Portugal.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3234152.3234187

1 INTRODUCTION

Software development organizations are hard pressed to meet the increasing demand for secure software [12, 19, 38]. 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 [8], 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 [5]. 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 [22, 43]. 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. 35].

The agile methods improve productivity by narrowing the scope of implementation into specific features within a fixed time frame [e.g. 1]. Focused development allows for meticulous concentration on the quality and functionality selected into the iteration backlog. By selection of the tasks, the team and the customer can be reasonably assured that the work is done in order to achieve the objectives currently considered most important for the software product under development. The differences between the methodologies have been categorized : the methods are determined to be either risk-driven or value-driven [11]; hybrid models, such as Disciplined Agile Delivery [2], 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 combined with software security engineering activities?

This question is considered primarily from the viewpoint of agile software engineering in the following chapters. 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 industry[25];

XP '18 Companion, May 21–25, 2018, Porto, Portugal

ACM ISBN 978-1-4503-6422-5/18/05…$15.00
https://doi.org/10.1145/3234152.3234187

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 16]. 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 [9]. Mainstream software development methods are extremely value-focused, and as such perform poorly when facing *non-functional* requirements [32]. 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 [41], and even suggestions that agile methods are inherently ill-suited to produce secure software [cf. 33].

Security standards are guidelines for security implementation, and several international software security regulation frameworks exist. The Common Criteria [23] 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 [23, 24].

Agile software development is characterized by a *light-weight* management model: pre-planning is minimized, teams and developers have a high degree of autonomy and the development of the software resource itself is the primary target. Agile, as a descriptive adjective itself, is a combination of *values* and *principles* first expressed in The Agile Manifesto. Lightweightedness, in regard to software security engineering, is particularly reflected in principle #11, which states *"[t]he best architectures, requirements, and designs emerge from self-organizing teams."*

This particular statement has been strongly criticized by well-known proponents of software architecture, such as Bellomo et al. [7]. However, despite so named, in an analysis by Séguin et al. [37] it was noted that this particular agile principle does not qualify as a *software engineering* principle: it does not contain a prescriptive statement, it is not *testable*, nor are its *consequences observable*. Using these conspicuous statements as excuses to exclude the whole mainstream software development methodologies from being viable to produce secure software appears to reflect a rather poor understanding of both security engineering and agile software development methods. *Skill* acquired through security training and experience in security engineering is projected to have a remedial effect [30].

2.1 SSE models in an agile context

SSE is predominantly performed by sequential models. 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.

Security engineering is an established tradition, with earliest formalized guidelines stemming from the United States Department of Defence in the 1980s [17]. The highly formalized and mechanistic process of evaluating a system's security needs was performed by applying specific evaluation criteria [18] into pre-classified data and pre-classified users, resulting in predefined requirements of security functionality and security assurance. This approach was then combined with several other standards and developed into a 'de-militarized' version today known as The Common Criteria [14, 23].

Early computer security was entirely dependant on approved and certified security products [45]. Government regulation eventually mandated the use of security verification and assurance to prevent misuse and tampering of electronic data. Security assurance, originally meant to consist mostly of electronically produced data (i.e., various security logs) and basic functionality descriptions quickly extended into a complex set of security policies and even formalization requirements for the software products implementation itself (see e.g. Security Policy Models in [23]).

At least two common maturity models address also development-time information security issues: (1) The Software Security Engineering Capability Maturity Model (SSE-CMM) is the ISO/IEC's heavily process-oriented security management, metrics and implementation framework [24]. This model originates from the Capability Maturity Model Integration, developed by the Carnegie Mellon University and currently maintained by the CMMI Institute [26]. As a process model, applying the CMM-based model can be very costly [15], 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) [39]. 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 [29], 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.

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 [36], currently the most commonly used mainstream software development methodology [34, 42]. 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

Table 1: Selected agile practises, processes and artifacts and their use as reported in [25]

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.

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 [4] and Ayalew et al. [3]. 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.

The source survey for Table 1 has also been reported as agile practises' significance in reducing technical debt [21]. 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.

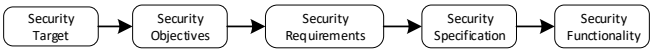


Figure 1: Simplified software security target framework (adapted from the Common Criteria [23]).

3 SECURITY ACTIVITIES IN AGILE SOFTWARE DEVELOPMENT PROCESS

This section lists all the security activities extracted from Microsoft SDL [22, 27], The Common Criteria [14, 23] and OWASP SAMM [28]. The activities for each lifecycle phase are examined and their adaptability to agile development and a matching agile activity are presented.

3.1 Pre-requirement

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

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

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 [10] as well as security engineering [see e.g. 30].

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. 30].

3.2 Requirement

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.

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

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

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 only agile activity directly addressing requirement elicitation and prioritization is the Planning Game [6]. 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. [40], 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 [44] are used to review the requirements and transform them into implementable features and functionality.

3.3 Design

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.

3.4 Implementation

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.

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

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.

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

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 [31]; pair programming also acts as a substitute for formal reviews [13]. 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

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

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

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

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.

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

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.

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 [20]. 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:

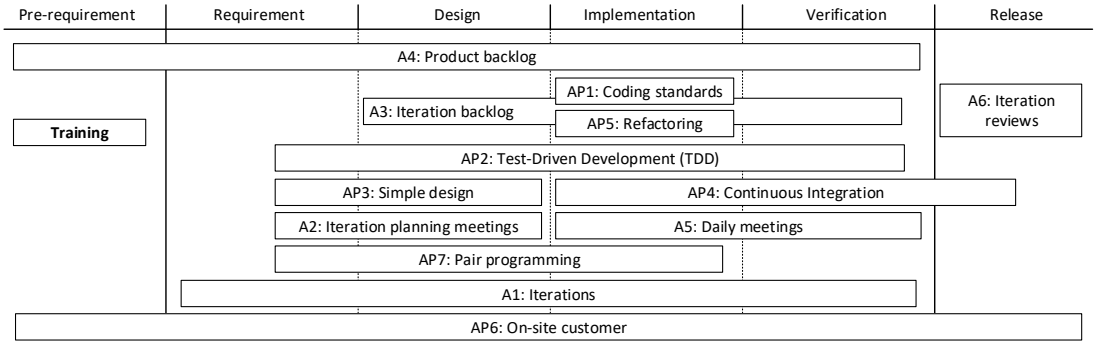


Figure 2: Agile activities mapped into the security development life cycle.

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.

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

[2] 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.

[3] 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

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

[5] 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.

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

[7] Stephany Bellomo, Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. 2014. How to Agilely Architect an Agile Architecture. *Cutter IT Journal* 27, 2 (2014), 12–17.

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

[9] 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

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

[11] 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

[12] 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

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

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

[15] 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.

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

[17] DoD. 1983. *TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA.* United States Department of Defence.

[18] DoD. 1985. *GUIDANCE FOR APPLYING THE DEPARTMENT OF DEFENSE TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA IN SPECIFIC ENVIRONMENTS.* United States Department of Defence.

[19] 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

[20] 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

[21] 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

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

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

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

[25] 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

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

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

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

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

[30] 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

[31] 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

[32] 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

[33] 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

[34] 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

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

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

[37] Normand Séguin, Guy Tremblay, and Houda Bagane. 2012. Agile Principles as Software Engineering Principles: An Analysis. In *Agile Processes in Software Engineering and Extreme Programming*, Claes Wohlin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–15.

[38] 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

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

[40] 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

[41] 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.

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

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

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

[45] J. R. Yost. 2015. The Origin and Early History of the Computer Security Software Products Industry. *IEEE Annals of the History of Computing* 37, 2 (Apr 2015), 46–58. https://doi.org/10.1109/MAHC.2015.21

Paper III

Securing Scrum for VAHTI

Rindell, Kalle and Hyrynsalmi, Sami and Leppänen, Ville (2015). In *Proceedings of 14th Symposium on Programming Languages and Software Tools*, pages 236–250.

Securing Scrum for VAHTI

Kalle Rindell, Sami Hyrynsalmi, Ville Leppänen

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

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

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

1 Introduction

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

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

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

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

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

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

2 Motivation, Research Questions and Related Work

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

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

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

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

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

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

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

3 Scrum

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

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

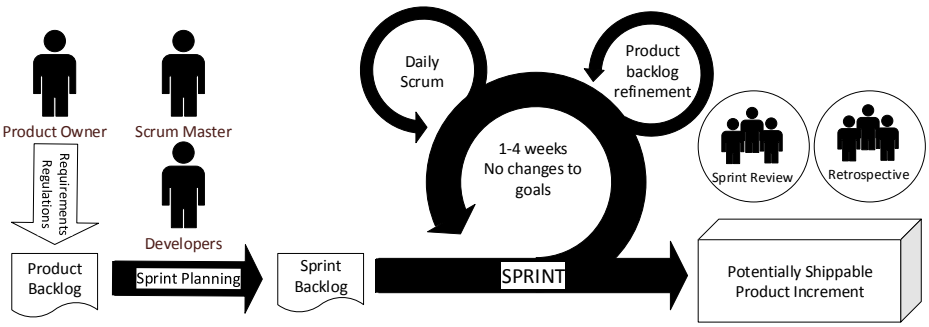


Fig. 1. The Scrum process (adapted from [6])

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

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

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

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

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

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

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

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

4 VAHTI instructions for application development

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

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

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

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

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

Table 1. VAHTI requirements for development method per security level

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

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

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

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

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

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

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

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

5 VAHTI-Scrum

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

planned releases, if the organization has decided to use those. It is also recommendable to arrange a security sprint, 'sprint zero', before commencing the actual implementation; alternatively, some security work may be completed as security spikes. Spikes could occur at any time - mainly in the early phases of the project, so they do not have even approximate placement on the project time line. In similar fashion, the hardening sprints may be inserted between the implementation sprints, to prepare for audits or just enhance security. Especially on heightened and high levels the amount of security work can be so big that a separate sprint is justified. It is important, though, that security elements become integral part of all the team's work: all tasks should include some security considerations, at least in the form of risk analysis, just as all sprints contain security tasks.
Figure 2 presents the modified Scrum process with the additional requirement sources, roles, security related actions during the sprints and the artifact types produced by the security processes.

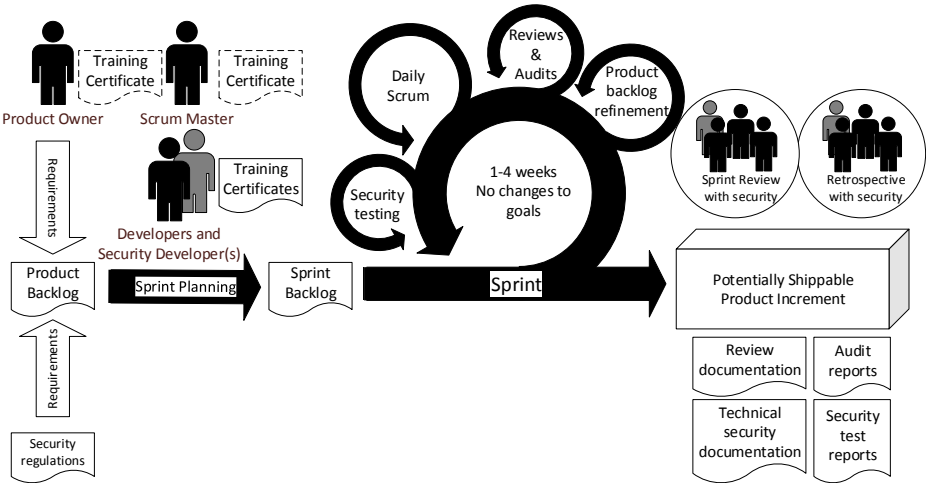


Fig. 2. VAHTI compliant Scrum process

The modifications to standard Scrum are listed below:

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

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

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

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

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

5.1 Roles

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

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

5.2 Tasks

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

Basic level

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

Heightened level

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

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

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

High level

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

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

5.3 Artifacts

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

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

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

6 Conclusions and future research

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

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

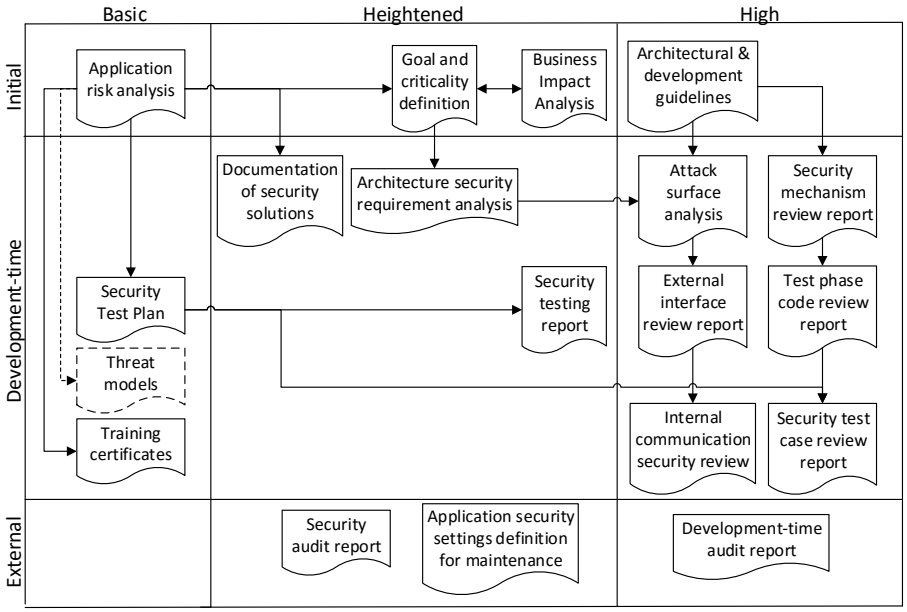


Fig. 3. VAHTI documentation artifacts

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

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

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

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

References

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

Paper IV

IV

Surveying Secure Software Development Practices in Finland

Rindell, Kalle and Ruohonen, Jukka and Hyrynsalmi, Sami (2018). In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 6:1–6:7.

Surveying Secure Software Development Practices in Finland

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

Jukka Ruohonen
University of Turku
Turku, Finland
jukka.ruohonen@utu.fi

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

ABSTRACT

Combining security engineering and software engineering is shaping the software development processes and shifting the emphasis of information security from the operation environment into the main information asset: the software itself. To protect software and data assets, software development is subjected to an increasing amount of external regulation and organizational security requirements. To fulfill these requirements, the practitioners producing secure software have plenty of models, guidelines, standards and security instructions to follow, but very little scientific knowledge about effectiveness of the security they take.

In this paper, we present the current state of security engineering surveys and present results from our industrial survey ($n = 62$) performed in early 2018. The survey was conducted among selected software and security professionals employed by a selected set of 303 Finnish software companies. Results are compared to a commercial survey, the BSIMM version 8 and the similarities and distinct differences are discussed. Also, an analysis of the composition of security development life cycle models is presented, suggesting regulation to be the driving force behind security engineering in software industry.

KEYWORDS

software engineering, agile, security engineering, survey

ACM Reference Format:
Kalle Rindell, Jukka Ruohonen, and Sami Hyrynsalmi. 2018. Surveying Secure Software Development Practices in Finland. In *ARES 2018: International Conference on Availability, Reliability and Security, August 27–30, 2018, Hamburg, Germany*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3230833.3233274

1 INTRODUCTION

Information security and privacy of personally identifiable information are main concerns of all organizations involved with processing or storing digital data. *Security engineering*—the systematic effort to produce more secure computing environments [1]—is primarily concerned with the protection of operating environments and the post-development phases in the software life cycles. The overlapping of security engineering with software engineering has been nominal and more or less an afterthought. Security engineering covers external reviews, pre-scheduled milestones, and formal certifications, among other related processes. By and large, these do not contribute to the primary business requirements of software.

Several maturity frameworks, software life cycle models, and security standards have been established. The goal of these is to guide and commensurate software development processes towards producing secure software systems that meet formal security requirements with adequate security assurance. These include the Open Web Alliance Security Project's (OWASP) Software Assurance Maturity Model (SAMM) [17], the Building Security in Maturity Model (BSIMM) [13, 26], a generic design-oriented life cycle model from Synopsys, and the Security Development Lifecycle (SDL) from Microsoft [6, 14]. Of these, only BSIMM claims to be based on empirical observations by being created by observing and analyzing real-world data from leading software security initiatives.

There is a lack of scholarly empirical research on BSIMM, and to the best of our knowledge, this paper is the first to examine BSIMM from a scholarly viewpoint. This study's research objective is to evaluate BSIMM and Microsoft SDL against the security and software engineering practices currently used in the Finnish software industry, and contributes to research on the linkages between software and security engineering.

To this end, we conducted a survey among software professionals in selected software companies in Finland and asked for their security practises. The survey was conducted by presenting a comprehensive list of software security engineering activities to the survey respondents. The respondents were asked to rate these based on their usage, and to estimate the perceived effectiveness of these activities, and their impact upon security. The rest of the paper is structured as follows: section 2 briefly reviews the related work. Results are presented in Section 3 and discussed in Section 4. Conclusions follow in the final Section 5.

2 BACKGROUND

2.1 Secure Software Engineering

Software engineering has primarily been concerned with the effectiveness of software development and the means to effectively create features and functionality that fulfill the software's primary 'business' objectives [7, 8]. At the risk of over-generalization, in the grand scheme of things, security has been traditionally seen as a secondary objective that falls to the parent field of computer science [3]. Traditionally security engineering was seen as a part of system engineering, a more technical and non-theoretical subject primarily concerned with the task of protecting software systems from security threats. If software security was recognized as an explicit goal, it was seen as a subdivision of software quality improvement efforts. The actual software security was introduced after a software product had already been released and deployed. The

ARES 2018, August 27–30, 2018, Hamburg, Germany

ACM ISBN 978-1-4503-6448-5/18/08…$15.00
https://doi.org/10.1145/3230833.3233274

primary goal was to protect the operating environment, whether computers or networks, by guaranteeing confidentiality, integrity, and availability.

Introduction of maturity models, such as ISO/IEC SSE-CMM [9], in the 1990s, formalized a systematic approach towards security integrated into the software development processes. However, since the introduction of these quality and security improvement efforts, software development processes have undergone a comprehensive paradigm change. Managing software project and its risks is no longer based on careful pre-planning and evaluation (and likely rejection) of change requests, but rather adapting to the change [2]. Another prominent trend in software development methodologies is concentration on the functionality, or 'value'. In this frame of thinking, security requirements are prone to gaining less attention, or even being dismissed as secondary. In these agile and lean terms, anything that is not recognized as producing value, is treated as *waste* [18]. Tendency to disregard, possibly without even recognition or understanding of the security requirements has proven to be a serious challenge to software security engineering. The paradigm change, starting in the mid-1990s, coincided with the vast increase in Internet-connected programmable devices and complexity of the software. This was prominently demonstrated by Microsoft, whose Windows 95, 98 and XP operating systems suffered from an enormous amount of security flaws and vulnerabilities [6].

This lack in software security was recognized and partially remedied by new efforts to mix software engineering and security engineering at the development phase. A substantial early contribution was the introduction of Security Development Lifecycle (SDLC) by Viega and McGraw in 2002 [26]. Microsoft, learning from their failures, also published a design-centered SDLC around mid-2000s [6]. McGraw and partners later went on to introduce the Building Security In Maturity Model (BSIMM) in 2009. This model is based on industry surveys, and provides a set of best practises for software security development and organizational security processes.

2.2 Related surveys

Stavru [23] has pointed out certain shortcomings in industry surveys in the field of software engineering. The key points of their analytic framework are here applied to the BSIMM. First of all, we do not know *who answered the survey*. The BSIMM does provide industry verticals (business areas where the participant companies belong) and good demographics about them, but does not clearly state who answered and what was her role. There is a mention of 109 executives and "more than 80 individuals", but no details are provided about details. Nor is it clearly stated *what were the questions* asked. To overcome these shortcomings, our survey adheres to the guidelines and good practises presented in [23].

The background information states that the companies have had a Software Security Group (SSG) initiative in place for about four years. The BSIMM does good job in providing links of how the SSG should be constructed, what kind of experts it should contain, and how it should be led, but tells very little of individual SSGs. The median size of the group is stated to be 5 (smallest 1, largest 130), and average size is 11.9 – but on top of that, all that we know is their average age and that there are 109 of them.

Finally, BSIMM describes itself as descriptive, not prescriptive. In other words, BSIMM provides a balanced scorecard for consulting—the winners and losers of a "software security popularity contest". In BSIMM, security activities are divided into four main domains: governance, intelligence, SDLC Touchpoints, and deployment. The focus in our survey is in SDLC Touchpoints, introduced in McGraw's 2006 praiseworthy book on software security [11]; its relationship to BSIMM is further explained by McGraw in [12]. Touchpoints are SSDL practises consisting of architecture analysis, code reviews and security testing, the building blocks of software security engineering. From a software engineer's point of view this contains the development-time activities and shapes the software security engineering processes and activities. There are, however, activities involved in all four domains of BSIMM in our survey – some directly worded after the BSIMM activities, some closely related. Based on our results, however, BSIMM appears to provide a somewhat biased view on software security. This is likely due to the simplified way they have chosen to report the usage of software security activities in their survey. Also the choice of respondents is an important factor; unlike BSIMM, our survey targeted software developers in general, whereas BSIMM appears to have a strong focus on security-oriented companies, and specifically on executives.

Software development practises in general have been extensively surveyed in Finland [21] and elsewhere [10]. Security education and security skills have recently been charted in Finland [22]. On the other hand, surveys about security software development process appear almost nonexistent. Besides BSIMM's surveys and reports, there exists very little evidence on *how*—if at all—software security is actually implemented during software development. This gap is unfortunate (and somewhat odd) compared to the substantial body of empirical research on software development processes gathered over the decades of its existence. To fill this gap, and to provide comparability with the previous studies, our survey, labeled "Secure Agile Survey", was aimed to confirm the software development practises in combination of 40 suggested development-time security activities.

3 RESULTS

The results are disseminated by first discussing the survey design and the activities observed. After this brief discussion, the main empirical points are delivered by focusing on the reported use of the activities and their perceived impact on software security.

3.1 Materials

The results presented are a part of a larger web survey on security engineering practices and agile software development processes currently used in Finland. The survey is structured into three main categories: (a) the background of the respondents and their organizations; (b) the typical agile software development processes used in the organizations; and (c) the security engineering practices used. All of these categories are linked. In particular, respondents were instructed to consider only those software projects that were implemented by using both agile and security engineering processes. While keeping this point in mind, this paper focuses on a subset of the security engineering practices.

Table 1: Security activities in the survey.

Phase	Source	Activity
Requirement		
	Other	Identify user roles and requirements
	SDL	Establish security requirements
	BSIMM	Create a data classification scheme and inventory
	BSIMM	Security requirement review
	SDL	Set quality gates
	BSIMM	Translate compliance constraints to requirements
	VAHTI	Define application goal and criticality
	SDL, VAHTI	Application's security and privacy risk analysis
	VAHTI	Business impact analysis
Design		
	SDL, VAHTI, BSIMM	Threat modeling
	SDL	Design requirements established
	Other	Abuse or misuse cases
	VAHTI	Architecture and Application Development Guidelines
	VAHTI	Application security configurations specified
	VAHTI, SDL	Attack surface analysis and reduction
	VAHTI	Application Security Settings Definitions
Implementation		
	BSIMM, SDL	Use coding standards
	SDL	Approved tools
	SDL	Static analysis
	SDL	Deprecation of unsafe functions
	Other	Security specific hardening sprints
	VAHTI	External interface review
	BSIMM	Use automated tools along with manual reviews
	VAHTI	Documentation of security solutions
Verification and Validation		
	*VAHTI	Security specific test cases
	SDL	Fuzz testing
	*BSIMM	Penetration testing
	SDL	Dynamic analysis
	SDL	Attack surface review
	VAHTI	Review security testing plans
	VAHTI	Code reviews during testing
	VAHTI	Automated testing tools
Release		
	BSIMM	Code signing
	SDL	Incident response plan created
	BSIMM	Documentation required by regulations
	*VAHTI	Internal security audits
	*VAHTI	External security audits
	SDL	Formal certification
	*VAHTI	Security patch planning

Items marked with * are differently worded than corresponding BSIMM activities.

The initial questionnaire was piloted with a small set of professional security engineers and researchers. Based on their feedback, the draft questionnaire was adjusted regarding questions that were found to be hard to understand. After this iterative development, the questionnaire was posted in December 2017 via email to several Finnish software companies and engineers working in these companies. In addition, the survey was promoted online by asking people to forward it to interested parties. This promotion included also social media platforms. After a month, reminder messages were sent. The questionnaire was closed in late January 2018. In total, 62 usable responses were received.

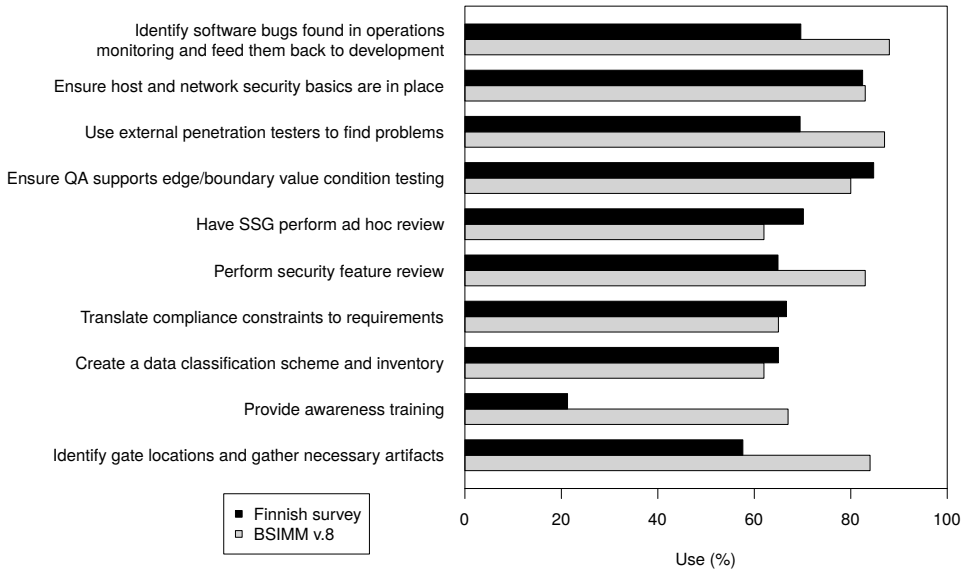


Figure 1: A Comparison of Selected Security Engineering Activities

3.2 Activities in the Survey

The surveyed security engineering activities, presented in Table 1 were drawn from established models and standards, with few additions drawn from extant literature. However, for comparing the results to BSIMM, a number of activities were explicitly included from the BSIMM version 8 [13]. The other activities surveyed were based on the Microsoft's SDL and the Finnish governmental VAHTI framework. The inclusion of VAHTI is particularly important in Finland, given the country's extensive public sector, currently undergoing a large-scale digitalization process (see e.g. [15]). VAHTI instructions for software development have been created by security professionals by collecting best practices and security activities [25].

The BSIMM promotes a "top-12" list of organizational security activities, which are expressed as "activities that 'everybody' does". According to BSIMM's argument, a security practise can be formed by selecting three activities from each of their four domains; one of these domains is a security development lifecycle model, the SDLC Touchpoints which resides directly within the scope of our survey. Microsoft does not explicitly state the criteria how the activities were selected into the SDL. the model has not received any significant changes since its creation. The focus at Microsoft appears to have been simplicity, scalability and applicability to various organizations and products [6]. VAHTI instructions for software development contain a comprehensive set of security activities, largely based on best practises; the usage and impact of VAHTI instructions for software development has been analysed in [20]. The activities in these models have a significant overlap, mainly due to the generic nature of the Microsoft SDL. BSIMM claims to be descriptive whereas VAHTI is prescriptive to the level of a *de facto* national standard.

Our survey was activity and lifecycle based and the questions were grouped by the SDLC phases. Primary purpose was measuring their usage, and for activities they had used, the respondents were asked also for an evaluation of their effectiveness. The 40 included security activities were divided into five development phases: requirement elicitation, design, implementation, verification and release. Of the 40 activities five were word-for-word picked from the BSIMM top 12 using their distinctive nomenclature. Further five were included in or similar to the activities picked from SDL or VAHTI. As we concentrated on development-time activities, user training was asked giving a yes/no option regarding the security training given by their current organization; the results may not be directly comparable.

Activities of the SDL were also prominently featured, with 11 of SDL's 15 development-time activities selected, added with the training activity with the same notes as above. Rest of the activities are selected from VAHTI, complemented with three common security activities, selected from software security engineering literature as reviewed in [19]. These activities are labeled as 'Other' in Table 1. As security activities are rather universal, there was also some overlap with certain activities stemming from multiple sources. As usage data of the other models does not exist, our comparison is necessarily specific to BSIMM.

3.3 Actual Use

Most of the ten BSIMM-related activities examined were surveyed with a Likert-scale. The only exception is a question about security awareness and training provided by the respondents' employers. This question was asked with a dichotomous scale: either security training was provided or it was not provided. A five-level scale

was used for the other activities: a given activity was used systematically, mostly, sometimes, rarely, or never. For these activities, the respondents were asked to frame their answers with respect to those software projects that used both agile processes and security engineering practices. The selected activities are the ones in the Finnish study overlapping with the BSIMM' top-12 practices, directly available for comparison.

For comparing the survey results and the usage frequencies collected from the recent BSIMM report [13], all survey answers that indicated at least some usage were collapsed into a single category. For each activity, the missing values were subsequently removed. The results are summarized in Fig. 1. The comparison should be interpreted only tentatively due to various methodological reasons, starting from the lack of details on how the BSIMM reports are assembled empirically. While keeping this remark in mind, most of the ten activities align even surprisingly well between the Finnish companies surveyed and the global software companies participating in BSIMM. However, there is a notable difference in terms of the security awareness and training programs provided in the companies—only about 16% of the survey respondents reported to have received formal security training from their employers. This observation is in stark contrast to the BSIMM results. To a lesser extent, there is a difference also in terms of identifying so-called gate locations such as milestones and other release engineering aspects. For all remaining activities, the usage can be reasonably interpreted to be similar, given the comparison's implicit but presumably wide error margin.

Another way to look at the use of the activities is to examine how frequently the activities were used. The original Likert-scales are suitable for this task. The results are summarized in Fig. 2, sans the training question with its dichotomous scale. The results are again rather similar between the ten activities with one exception. When compared to the other eight activities shown in the figure, ensuring the fundamentals of host and network security was used much more systematically. This result is not surprising. After all, doing "*software security before network security is like putting on your pants before putting on your underwear*" [13]. The Finnish respondents seem to agree with this truism.

3.4 Perceived Impact

The use of a particular security engineering activity does not mean that the activity would be particularly important for improving security. For this reason, the survey respondents were also asked to evaluate the impact of each activity upon security. As a survey cannot answer to a question about actual security of a software system, the term impact should be understood as the respondents' educated opinions on the *perceived impact* upon security. Such perceived impact was again solicited with a five-level Likert-scale, ranging from a very high impact to a very low perceived impact.

There are three noteworthy observations to make from the results summarized in Fig. 3. First and foremost: for all nine activities, the perceived impact is much higher than the corresponding use (cf. Fig. 2). In fact, all of the activities are perceived to have at least some impact upon security. Only a negligible amount of answers fall into the category of very low perceived impact. Second, the fundamental premises of host and network security match in terms of use and perceived impact. Third, penetration testing is perceived to have a very high impact, although this activity is only seldom systematically used in Finland. Excluding this interesting detail, the results largely bespeak about a mismatch between use and impact. In other words, there is still much to improve. A possible explanation for this mismatch relates to regulations and standards.

4 DISCUSSION

As a software development objective, security is typically implemented in a way that provides various types of *assurance* [24], by which it is then evaluated. Most of the software security engineering activities provide means to that end: reviews, tests, verification, and extensive documentation produce security assurance artifacts and act as evidence of security existence. The original idea behind security assurance was to prove the existence of such security mechanisms that enforce the system's security policies; in early security specifications this meant primarily programmatic evidence, i.e., machine-produced log files [4]. Over several iterations of regulation, the definition of assurance expanded to cover also various reviews and programmer-created documentation [5].

This trend is notable in both BSIMM and the Finnish survey: of the nine common activities, only two are direct security improvement activities: "Identify software bugs found in operations monitoring and feed them back to development" (BSIMM code CMVM1.2), and "Ensure host and network security basics are in place" (BSIMM code SE1.2). In the software development lifecycle, both these activities belong to phases that take place *after* development. It should be well noted that feeding the found defects from maintenance phase into the development backlog is a vital security activity in DevOps model and links the maintenance phase directly back to the design, implementation and verification phases.

The rest of the activities fall into category of security assurance: two of the activities belong into domain of security testing: Ensure QA supports edge/boundary value condition testing (BSIMM code ST1.1) and Use external penetration testers to find problems (BSIMM code PT1.1). Notably, BSIMM promotes the penetration testing activities as separate from other security testing.

The rest of the activities are various reviews. At code level, expert reviews are used to enforce coding standards and locate security design flaws and bugs; reviews are also applied to security architecture and security features. Reviews are also perceived very efficient and cost-effective ways to improve software security [24]. The main issue with reviews is that they are performed typically by external personnel, which may not be available unless explicitly required – and paid for – by the customer. This leads to two minor conclusions about the BSIMM: first, it appears regulation-driven with emphasis on security assurance and compliance requirements; second, it notably promotes use of external security experts despite one of BSIMM's basic elements is the organization's own SSG, or a "security satellite".

5 CONCLUSIONS

The foremost conclusion is clear: (a) the security engineering activities currently used in Finland align well with the BSIMM-based activities used in the global software industry. The reasons for this alignment are likely also similar. Despite increased promotion for

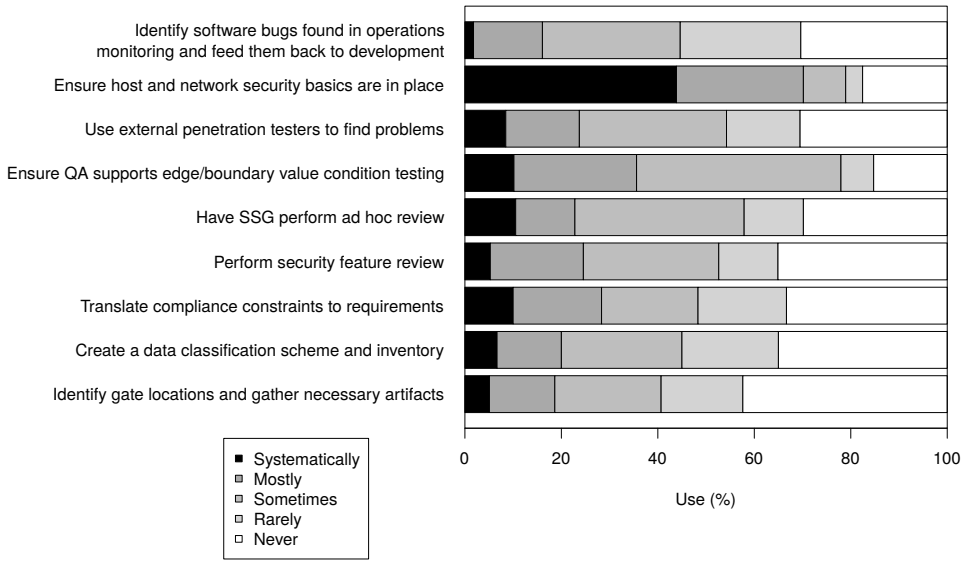


Figure 2: A Breakdown on the Use of Selected Security Engineering Activities

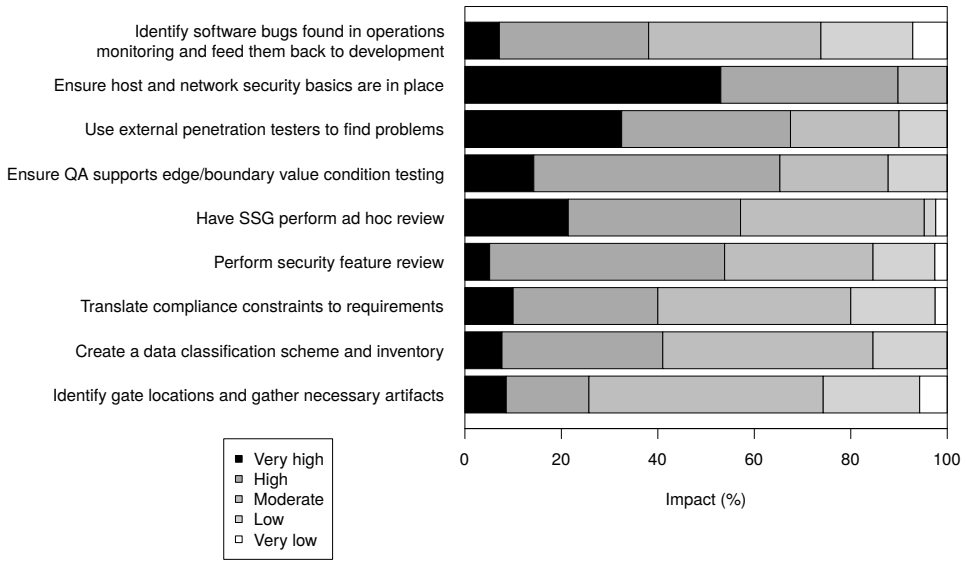


Figure 3: A Breakdown on the Perceived Impact of Selected Security Engineering Activities

security and increased recognition as an important property of good software, security engineering in software development appears still to be driven by regulation. That is, the rationale for using the majority of "top-12" activities surveyed can be plausibly explained by regulation.

The results further align rather well with a recent industry survey according to which setting standards is often more important than actually following through, external reviews are seen as important for security, and last but not least, many companies fail to perform desired security activities [16]. In particular, (b) the small set of activities surveyed are actively used in Finland, but the use is still limited compared to the perceived impact upon security. When compared to BSIMM's surveys, (c) security training is only seldom used according to the results. This observation may relate to a selection bias: according to the background information collected, most of the respondents are professionals with more than six years of software development experience. As all respondents have also explicitly worked in the domain of security engineering, formal security training may be a redundant activity for the majority of the respondents. Finally, (d) it is worth pointing out that penetration testing was perceived as particularly important for security. This result is in contrast to the mentioned survey, which rather pointed out the limitations of penetration testing when compared to code reviews [16]. Given that penetration testing is not widely used in Finland according to the results, the reason may relate to the concept of perceived impact on software security. In other words, an activity that is widely promoted by consultants and industry associations may not correlate with the actual impact of the activity. This remark leads to a couple of important points for both research and practice.

First, measuring the popularity of security activities hardly constitutes a maturity model. Instead of just providing a ranking, a maturity model should describe a tangible framework for organizations to translate their security objectives and requirements into a set measurable security activities, processes, and artifacts. Generic security models, to-do-lists, and prescriptive processes are necessary when the security objectives of the software development process, and thus the software product, are based on security regulation, laws or standards.

Second, in most software projects, security engineering activities should be arguably based on the project's threat models and risk assessments, which depend on the application area, implementation platform, and the operating environment. Instead of prescribing a SSE process, the SSDLs would be more beneficial when they contain a set of targeted security engineering activities. From these sets, the software security experts would be able to pick the most effective ones fitting their software development processes and fitting their specific security profiles.

Software security is tightly related to the privacy of the customers of an electronic business, and increasingly also all citizens in an information society. Regulations will continue to drive software security by providing a set of security objectives, but it is in the hands of software and security practitioners to define and implement the correct and most effective measures to achieve those objectives. In software industry, this means setting up processes that integrate into the development processes as seamlessly and effortlessly as possible; optimally in the way that the benefit gained from the security measures exceeds the cost. Building such a framework should be based on universal and generic building blocks and theoretical constructs. Further empirical research can suggest how to put these together in a way that is beneficial in practice.

REFERENCES

[1] Ross J. Anderson. 2008. *Security Engineering: A Guide to Building Dependable Distributed Systems* (2 ed.). Wiley Publishing.

[2] K. Beck. 1999. Embracing change with extreme programming. *Computer* 32, 10 (Oct 1999), 70–77. https://doi.org/10.1109/2.796139

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

[4] DoD. 1983. *TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA.* United States Department of Defence.

[5] DoD. 1994. *SOFTWARE DEVELOPMENT AND DOCUMENTATION.* United States Department of Defence.

[6] Michael Howard and Steve Lipner. 2006. *The security development lifecycle.* Vol. 8. Microsoft Press Redmond.

[7] IEEE. 1990. *IEEE Standard Glossary of Software Engineering Terminology.* 1–84 pages. https://doi.org/10.1109/IEEESTD.1990.101064

[8] ISO/IEC. 2001. *ISO/IEC 9126. Software engineering – Product quality.* ISO/IEC.

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

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

[11] Gary McGraw. 2006. *Software Security: Building Security In.* Addison-Wesley Professional.

[12] Gary McGraw. 2012. Software Security. *Datenschutz und Datensicherheit - DuD* 36, 9 (01 Sep 2012), 662–665. https://doi.org/10.1007/s11623-012-0222-3

[13] Gary McGraw, Sammy Migues, and Jacob West. 2017. *Building Security In Maturity Model (BSIMM), version 8.* Technical Report. BSIMM.

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

[15] OECD. 2018. Government at a Glance 2017 – Finland Country Fact Sheet. (2018). https://www.oecd.org/gov/gov-at-a-glance-2017-finland.pdf

[16] Andy Oram. 2017. The Alarming State of Secure Coding Neglect: A Survey Reveals a Deep Divide Between Developer Aspirations for Security and Organizational Practices. (2017). O'Reilly Media, Inc. Referenced in 5th of May 2018: https://www.oreilly.com/ideas/the-alarming-state-of-secure-coding-neglect.

[17] OWASP. 2017. Software Assurance Maturity Model. (2017). https://www.owasp.org/images/6/6f/SAMM_Core_V1-5_FINAL.pdf

[18] Mary Poppendieck and Tom Poppendieck. 2003. *Lean Software Development: An Agile Toolkit: An Agile Toolkit.* Addison-Wesley.

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

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

[21] 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

[22] Reijo M. Savola. 2017. Current Level of Cybersecurity Competence and Future Development: Case Finland. In *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings (ECSA '17).* ACM, New York, NY, USA, 121–124. https://doi.org/10.1145/3129790.3129804

[23] Stavros Stavru. 2014. A critical examination of recent industrial surveys on agile method usage. *Journal of Systems and Software* 94 (2014), 87 – 97. https://doi.org/10.1016/j.jss.2014.03.041

[24] Jose M. Such, Antonios Gouglidis, William Knowles, Gaurav Misra, and Awais Rashid. 2016. Information assurance techniques: Perceived cost effectiveness. *Computers & Security* 60 (2016), 117 – 133. https://doi.org/10.1016/j.cose.2016.03.009

[25] VAHTI 1/2013. 2013. Sovelluskehityksen tietoturvaohje. (2013). https://www.vahtiohje.fi/web/guest/vahti-1/2013-sovelluskehityksen-tietoturvaohje Referenced 8th Oct. 2017.

[26] John Viega and Gary McGraw. 2002. *Building Secure Software: How to Avoid Security Problems the Right Way* (1st ed.). Addison-Wesley.

Paper V

V

Case Study of Agile Security Engineering: Building Identity Management for a Government Agency

Kalle Rindell and Sami Hyrynsalmi and Ville Leppänen (2017). *International Journal of Secure Software Engineering*, 8(8):43-57

International Journal of Secure Software Engineering
Volume 8 • Issue 1 • January-March 2017

Case Study of Agile Security Engineering:
Building Identity Management for a Government Agency

Kalle Rindell, Informaatioteknologian laitos, University of Turku, Turku, Finland
Sami Hyrynsalmi, Tampere University of Technology, Pori, Finland
Ville Leppänen, Informaatioteknologian laitos, University of Turku, Turku, Finland

ABSTRACT

Security concerns are increasingly guiding both the design and processes of software-intensive product development. In certain environments, the development of the product requires special security arrangements for development processes, product release, maintenance and hosting, and specific security-oriented processes and governance. Integrating the security engineering processes into agile development methods can have the effect of mitigating the agile methods' intended benefits. This article describes a case of a large ICT service provider building a secure identity management system for a sizable government agency. The project was a subject to strict security regulations due to the end product's critical role. The project was a multi-team, multi-site, standard-regulated security engineering and development work executed following the Scrum framework. The study reports the difficulties in combining security engineering with agile development, provides propositions to enhance Scrum for security engineering activities. Also, an evaluation of the effects of the security work on project cost presented.

KEYWORDS

Agile, Case Study, Infrastructure, Scrum, Security, Standard, VAHTI

1. INTRODUCTION

Security regulations are an important driver in various aspects of software development and information systems and services. Even in the cases when formal security standards or guidelines are not strictly required, the drive for security still guides the selection of design patterns and technological components, as well as the design and development work. Increasing diversity in development methods, technology, and the environments where the systems are used, have prompted organizations to follow various security standards, as well as created the need to establish new ones to guarantee adequate security assurance. In 2001, the government of Finland begun to issue a set of security regulations, called VAHTI instructions[1]. Compliance with the instructions is now mandatory for all government agencies, and the regulation is also applied to any information system and data connected to a VAHTI-classified system.

While the importance and use of security regulations has increased, the use of lightweight software development processes and methods, i.e., agile development, has become the *de facto* standard in the industry (VersionOne, 2016). While there exists a series of suggested methods how to conduct security engineering activities in an agile project (see e.g. Alnatheer, Gravel & Argles, 2010; Baca

DOI: 10.4018/IJSSE.2017010103

& Carlsson, 2011; Beznosov & Kruchten, 2004; Fitzgerald, Stol & Sullivan, 2013; Ge, Paige, Polack & Brooke, 2007; Pietikäinen & Röning, 2014; Rindell, Hyrynsalmi & Leppänen, 2015), the empiric evidence is still largely anecdotal and the cases reported specific to an industry or a single company. The study reported in this paper is exploratory, and thus the research, by its nature, explorative. This study reports the experiences in agile development in a security regulated environment. The research objective (RO) is:

RO: Identify best practices as well as hindrances of using agile software development methodologies in security engineering.

The results contribute to the on-going discussion by being a result of a deep analysis of combining security engineering with an agile method in an industry setting. Furthermore, the result of this study pave the way for further work deepening our understanding on the benefits and drawbacks of using agile software development methodologies in security sensitive development work.

In the case described, a Scrum project was conducted with the objective of building an IDM system for VAHTI-compliant information systems, and a secure VAHTI-compliant server platform to host the systems, including the IDM. The server platform was to be used also to host software development projects (with certain dispensations). The project was executed during 2014 and 2015, and had a duration of 12 months. The development team was split into two to three geographically dispersed groups, with the actual amount of teams involved dependent on the tasks at hand and the overall phase of the project. As a standing practice with the government agency that initiated the building of the platform, the project was managed using unmodified "textbook version" of Scrum. This called for strict adherence to fixed-length sprints, well-communicated product and sprint backlogs and daily progress monitoring by the Product Owner and steering group. The project was under strict control of the Project Management Office, and schedules of related infrastructure and software development projects were depending on the results of this project. Compliance with VAHTI was a central objective of the project. In addition to VAHTI, the client agency had also their own additional security demands, as well as recommendations from other government agencies, most importantly the National Cyber Security Centre's (NCSA-FI)[2]. The server platform to be built was to be acceptable for use for all government agencies, as well as private companies or organizations requiring similar level of VAHTI compliance.

This paper presents how Scrum was applied for the security-related work required in the project, and how the project was conducted. As the study revealed that not all the objectives of using `pure' Scrum were not met, suggestions are made to improve the efficiency of the development work by introducing rudimentary security engineering extensions to the Scrum framework. The modifications include a new role for a security developer, and also suggest specific security sprints and other security-oriented additions to the run-of-the-mill Scrum. We also discuss how the introduction of the security engineering activities into the project affect cost, efficiency and the conduct of the project.

2. BACKGROUND AND MOTIVATION

The use of agile methods has become an industry practice, whereas the security standards regulating software development processes, such as ISO/IEC 21817 (2008) and ISO/IEC 27002 (2013) originate in the time preceding the agile methods. Based on the literature, and also the findings this observed case, the typical approach to agile security engineering is to simply start using the methodology at hand without formal adjustments, with the notable exception of thorough and formal approach to security engineering described by Baca & Carlsson (2011) and Fitzgerald & al. (2013). There are even well-documented cases of attempts to achieve formal ISO/IEC capability maturity level incorporating agile methods, such as Diaz, Garbajosa & Calvo-Manzano (2009). Unfortunately, the findings and

suggestions made in these studies were not directly applicable in a project that was not strictly restricted to software development. Instead, a more *ad hoc* approach was used. In this approach, the security-related tasks are treated simply as items in the backlog: the security requirement items are converted to tasks, given story points, and completed among the other items as best seen fit. When security items which cannot reasonably be time-boxed because of the inherent uncertainties of the work, or the inexperience of the team, they separated from the Scrum sprint cycle and completed in non-time-boxed spikes.

While the ad hoc method may succeed in achieving "minimum viable security" by complying with the formal requirements, it is hardly the most effective way to achieve the goals, nor does it provide the best security assurance for the end product. Achieving security assurance is by all means possible with careful planning, although lacking in proper security requirement management and security task pre-planning. Absence of these elements in the project management methodology tend to lead to inefficiencies and, consequently, delays and increased development costs. Lack of proper security assurance may also increase the amount and severity of the residual security risk during the software system's life span.

Our argument is that by adjusting the Scrum methodology to better align with security engineering tasks, the security cost overhead can be reduced while the security of the end product is enhanced, when compared to traditional sequential security engineering practices. This is achieved by incorporating the security processes into Scrum activities, as opposed to treating them merely as items in the backlog, by introducing new security-oriented roles into the development team. By incorporating the security engineering activities into the development method, the full benefit of incremental agile methods can be utilized to achieve better efficiency ratio and, arguably, better end products.

The next subsections provide more information about VAHTI, and the use of Scrum methodology in development projects requiring security standards compliance. Due to similarities in the requirements, the same observations and recommendations we make in this paper are found applicable also to software safety regulations, in e.g. medical field.

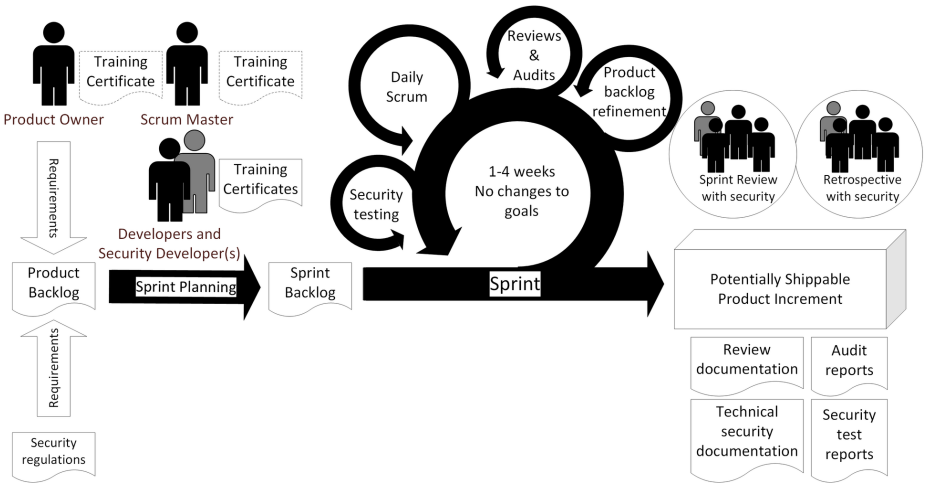
2.1. Security-Augmented Scrum

Scrum is a generic framework, originally intended to manage software development projects with small co-located teams. Scrum suggests that the product to be completed is divided into smaller components, or features, based on the customer requirements. These requirements are represented by user stories, which are then translated into product features by the development team. Features are then further divided into work packages or items, which are compiled into a product backlog. Items in the product backlog are completed in an incremental and iterative manner during short-term development sprints. The team, consisting of the Scrum Master, the Developers, and the Product Owner as customer's representative, determines the items to be completed during the next 2-4 weeks sprint, consisting of daily scrums. After the sprint, the work is demonstrated, and optionally the team performs self-assessment of the past sprint in a retrospect event.

In this representation the Scrum process is augmented by three major extensions, presented in Figure 1.

1. The role of a security developer. The security developer, or developers, focus on the security of the product, and typically create or review the documentation required to pass the security audits.
2. Security assurance provided by creating security artifacts, mostly security-related documentation. They consist of security training certificates required from the project team, but most importantly the architecture documentation, risk management plans, test plans, test reports, system's log files and other evidence required by the security auditor. The audits also produce reports, which are part of the security assurance provided for the customer.
3. Anticipation of and planning for security-related tasks. To better illustrate this aspect of security work, security engineering activities are presented as iterative tasks in the sprint cycle in addition

Figure 1. Security-oriented Scrum process and roles (adapted from Rindell, Hyrynsalmi & Leppänen (2015)



to the daily scrum. It should be noted that not all sprints may have all the security tasks, and if the organization decides to perform security-oriented security sprints, the daily scrum may entirely consist of security activities.

In a project using unmodified Scrum, such as the one used in this case, the security testing, reviews and audits are viewed as normal stories in the sprint backlog and executed as part of the daily scrum. In this view the security tests and audits are part of the product, as compliance with security standards and regulations is mandatory during development time. The main shortcoming is the difficulty or outright inability to estimate the amount of work involved in the security activities, which merits for giving them special treatment. By emphasizing the importance and special role of the security stories, compared to treating them as overhead and extra burden, is prospected to produce better results with higher efficiency. In effect, this will reduce the cost of the development work.

2.2. Security Regulations and Standards Applied

VAHTI is an open and free collection of the Finnish government's security guidelines, published on the Internet since 2001. The aim of this regulatory framework is to promote and enforce organizational information security, risk management and overall security competence of various government agencies, and harmonize security practices throughout the organizations. As of spring 2016, the collection comprises of 52 documents. The following VAHTI instructions were found to be relevant for this project:

- VAHTI 2/2009 "Provisions for ICT service interruptions and emergencies", FMoF (2009)
- VAHTI 2b/2012 "Requirements for ICT Contingency Planning", FMoF (2012)
- VAHTI 3/2012 "Instructions for Technical Environment Security", FMoF (2012)

Of these, only the document 2b/2012 is available in English. The other relevant documents are made available in Finnish, and their English titles translated for the purpose of this article. This also applies to much of the VAHTI terminology: official English translations may not exist, may be

inconsistent between documents or may change over time. As a curious example, the Finnish name of VAHTI board itself has changed recently, albeit the English translation has not.

In addition to the VAHTI requirements, the company responsible for building the platform is audited for compliance with ISO/IEC standards 9001, 27001, 27002, and 21817, as well as its own extensive management framework which it makes available for its clients for review. The company has functions in the United States, so also Sarbanes-Oxley (SOX) act applied. SOX is mostly concerned with the financial elements of the project, but still affected the work load of the Scrum Master by adding reporting responsibilities.

2.3. Data Security in VAHTI

VAHTI classifies the information systems into three security levels: basic, increased and high. The server platform, where the IDM system was installed, was built for the increased security level. Information contained in the systems is classified into levels from I to IV, where IV is the lowest. Information contained in a system audited for increased security level may contain clear-text information up to level III; in this case, however, all data was encrypted despite the official classification level.

2.4. About Security Engineering

The term `security engineering' in software industry comprises of all security-related tasks within a software-intensive product's life cycle. The standard's way to categorize these activities is to divide them into three main process areas: risk, engineering and assurance (see ISO/IEC 21817). The risk process assesses the risk and aims in minimizing it by assessing threats and vulnerabilities, and the impact they have, producing risk information. The security engineering process uses this information with other security-related input to define security needs and provides solutions to fill them. Assurance process collects and produces evidence of security's existence, and aims in its verification and validation. The ultimate goal of these processes is to identify and mitigate risk, and define the impact and actions to be taken when the residual or unrecognized risk is realized: what will happen when things break.

3. RESEARCH PROCESS

This study follows a case study design method by Yin (2003), and a qualitative research approach by Cresswell (2003). For the study, we were looking for a development project that was both using agile methods and fulfilling VAHTI regulations. Our decision was to focus on the VAHTI regulations, as they are viewed to be a national standard and, therefore the number of possible cases would be higher. In addition, we were looking for a project which would be either ended or near its ending in order that we would be able to evaluate the success of the used model. Finally, the selected case should be a representative candidate as well as be able to produce rich information about the phenomenon under study.

We ended up to select a project case where identity management and verification service was ordered by a governmental customer who required the use of VAHTI. The development work was done by following a modified version of Scrum software development method. As Scrum is currently one of the most used development methods, the findings from this case study should be representative.

The project was executed by a mature, well-known software product development and consultancy company in Finland. The company has a long history of both agile methods as well as producing information systems for the government. By the wish of the company, the client and the interviewees, all participants to the project shall remain anonymous.

For this study, we held a post-implementation group interview for the key personnel of the selected project. We used a semi-structured interview approach where time was given to the interviewees to

elaborate their thoughts about the phenomenon under study. The general questions concerned the scope and size of the project, amount of the personnel involved, and the daily routines of the team.

Also, the security standards that were applied to the project were gathered. The security mechanisms developed to implement the requirements were charted, along with how they were presented to the client and auditors. Finally, the amount of extra work caused by the security requirements was discussed and roughly estimated, and the interviewees recounted their views of the lessons learned in the project. The interview session also acted as a retrospective for the whole project, where the participants were able to express their views of positive and negative aspects of the project and the effect the security requirements had. The results of the interview were then analyzed by the researchers and the key observations were emphasized.

The project was selected as a potential research target due to its strict security requirement and the fact that it was executed and managed using Scrum framework. The interviewees were the Scrum Master and the head architect of the project. They were both deemed as key personnel of the project, and they were able to provide insight to the project background, its execution as well as its results. The selected interviewees were also the only ones that persistently participated in all of the sprints and were involved in the project for its whole duration.

The questions posed before the interviewees were divided into three groups. First three questions concerned the project background (Q1-Q3); following five questions concentrated on the project process, security standards, and feedback on the Scrum and security (Q4-Q8); and the final two questions canvassed the interviewees' views on the project results and success factors (Q9-Q10).

The questions were as follows:

[Q1:] Project subject and scope?
[Q2:] Project resources, budget, and duration?
[Q3:] Personnel locations, multi-site teams?
[Q4:] What VAHTI standards were followed?
[Q5:] What other security standards and regulation were included?
[Q6:] Other restrictions (safety, privacy, agency specific regulations)?
[Q7:] What types of steps were taken to enforce them?
[Q8:] How was the security assurance verified (audited) and audit trail maintained?
[Q9:] Did the budget and schedule hold, and what was the amount of extra work caused by security?
[Q10:] What were the lessons learned?

After the interview, some complementary questions were asked via emails to confirm certain details, but otherwise the initial interview session was deemed sufficient for the purpose of this study. Access to exact budget or workload figures, or system logs or other technical documentation was not made available for research: the security classification of the platform prevented using this data even for verification. Instead, the interviewees relied on their personal experience and notes made during the project, and provided best estimates on the matters in a general level accepted for publication.

4. CASE STUDY: THE PROJECT

The agency required a VAHTI compliant IDM platform for their various information systems, and for users and system administration and management purposes. The platform was to be built using off-the-shelf components, installed on common open source operating systems, and deployed onto a large scalable array of virtual servers. A similar IDM platform was built also to authenticate and manage the identities of the administrators who manage other VAHTI compliant servers and services, and is to be separately instantiated for regular office users as well based on the experience and solutions gained in this project.

The IDM was deemed to be a critical service in respect of agency's security, privacy and business requirements: while the agency had 650 internal users connecting to 450 separate server-side computer systems, they also manage a sizable array of contractors with a total user amount of up to 12,000. The building project was conducted at the same time the server platform itself was being built, which added to the challenge in such way that all the requirements of VAHTI were to be met by a novel implementation.

Nearly all the design and definition work was to be completed in this project. To add to the challenge, the work was to be performed using Scrum, mainly to ensure steering group's visibility to the project's progress, and also to enable reacting to any unexpected obstacles or hindrances met during the project execution. Unfortunately for the project team, the customer also saw use of Scrum as a method to change the project's scope during its execution by adding items to the product backlog, or removing them from there, which caused certain degree of confusion among the team and forced it to abandon some work already completed. These aspects of Scrum projects, however, are not a security issue but of a more generic field of project management, and therefore are not further discussed.

The development work consists of distinct phases, which were completed during one or more iterations:

1. **Definition:** synthesis of the requirements, component candidate selection, risk assessment and analysis.
2. **Design**: architecture design, definition of interfaces, component hardening plans.
3. **Development:** component research, modification (i.e., hardening), and installation.
4. **Testing, Reviews, Audits and Acceptance:** security testing, external audits and formal acceptance of the end product to be a part of the agency's system portfolio. In effect, security assurance processes.

As there were no formal milestones preset at the beginning of the project, the security gates, such as audits, were passed flexibly whenever each feature was considered to be mature enough. This removed certain amount of unnecessary overhead, as a traditional fixed milestone dates may call for the team to work overtime, which may get costly due to pay compensations and cause delays to other projects due to resource shortage.

4.1. Project Organization

The project involved an average of nine persons at any given time: a Scrum Master, a dedicated Product Owner, a Security Architect (in basic scrum, part of the development team in the role of a developer), and the developers split into their production teams based on location and occupation.

The service provider is a devout follower of ITIL[3], a well-established and recognized set of industry standard best practices for IT service management. Typically for an ITIL-oriented organization, the infrastructure production teams reside in their own "silos", with very little communication with other teams. Production teams were divided by their specialization, in this case `Storage and Backup', `Server Hardware', `Windows Operating Systems', `Linux Operating Systems', `UNIX Operating Systems', `Databases' and `Networks'. In addition, the IDM application specialists came from their own team, residing within a separate unit within the company. The project brought specialists from these various teams together, at least virtually --- for at least the daily 15-minute stand-up meeting. Due to team's multiple physically separated locations, the meetings were without exception held as telephone conferences.

The teams were utilized in different phases of the project in such way that only the Scrum Master, security developer (i.e., the architect) and the Product Owner had personal activities in every single sprint throughout the project. The developers were part of a larger resource pool, and drawn into the sprints or spikes in various phases of the project whenever their expertise was required.

4.2. Project Execution

Much of the work related to VAHTI regulations was done in the planning phase: it turned out that the client agency had compiled their own list of requirements, which was based on VAHTI but had new security elements added to the public requirements. This partially to compensate the dropping the specific requirements for VAHTI compliant application development (FMoF, 2013) in the beginning of the project.

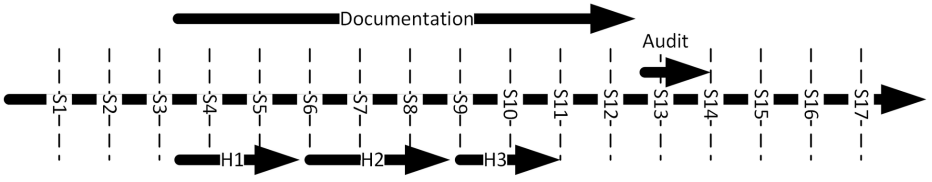
The project extended over a period of 12 months, from planning phase to accepted delivery of final sprint. The amount of work was measured in story points, and the average velocity of each sprint was 43 points. Divided with the average number of the developers (9) and the length of the sprint (15 work days) gives a rough estimate of a story point equaling three work days. As an overall measure, the story points give an impression of the size of the tasks. This sort of conversion may not be meaningful in general and outside of the scope of a single project, as the story points are primarily used to compare the features (or stories) to each other within a single project. For purposes of this study, the fact that largest single units of security work, the hardenings, were not performed in sprints and therefore not measured in story points, makes pinpointing the cost of security work much harder. In this case, the interviewees' estimates were the only source of the amount of workload, and although trusted to be reliable, exact figures would have been preferred.

From the beginning, the team's approach to the security tasks was pragmatic, although in terms of Scrum, rudimentary: stories that were found difficult to time-box at the time of their implementation, were taken out of the sprint cycle and completed as spikes. Prime examples of such tasks were operating system hardenings, a task essential for the platform security: the project team allocated resources to these tasks, and just ran them as long as the tasks took. This resulted in a project structure presented in Figure 2, where there were major side tracks to the main sprint cycle. As tasks such as these were in the very core of the project goals, it would have been beneficial to go through the trouble or even adjust the Scrum structure to better accommodate these items.

The sprints are represented as the main story line. The parallel lines represent the spikes that were executed outside the main sprint structure. Their results (deliverables) were demonstrated at a sprint demo, although they were executed independently without time-boxing. There were three distinct task types outside the sprint structure:

1. *System hardenings*, performed for each tier or environment of the system under development: Development, Quality Assurance (QA), and Production environments. The results obtained in the Development phase were not directly usable for the upper environments, whereas the QA environment was built to be production-like. As a result, the work done at QA phase was partly reusable at Production phase. Despite the technical similarities, the ITIL-guided maintenance models of these two environments were so great that the team proceeded in executing the Production environment hardenings as a spike as well.
2. *Documentation* was a ubiquitous process during the development. This included risk management, technical architecture and technical component documentation, test plans and

Figure 2. Project structure and spikes



reports. Documentation comprised most of the security assurance. Complete list of VAHTI requirements for documentation are presented in Appendix 3 of the VAHTI instruction 3/2012[4]. In this document, there are 224 mandatory requirements listed for the increased security level information systems. Almost all of these requirements call for some type of written evidence to be verified and reviewed, although most of the documentation artefacts are created in other than the development phase of the information system's life cycle.

3. *Reviews and audit* were performed based on the documentation and included physical testing of implementation.

4.2.1. Product Deployment Model

The demand for increased security (literally, the "increased level" on VAHTI security classification) also stated how the systems were deployed: to maintain audit trail, all changes to the production environment, including all server and hardware installations during its buildup, were performed following ITIL processes. These processes added extra levels of bureaucracy, and the team reported getting acceptance from the Change Advisory Board (CAB) for all changes to be made in the production environment had a very adverse effect on the deployment schedules. Combined with the policy of role separation between developers and maintenance personnel, this caused the building and installation of the production environment to be document-driven, bureaucratic and slow. The policy of separating the roles of developers and maintenance effectively prevents the `DevOps' type of continuous delivery maintenance model, and would require e.g. a form of "continuous security" model, such as presented by Fitzgerald & Stol (2014).

In this project, the continuous delivery model was used with the lower environments, speeding the rate of delivery significantly. When building the production environment, the flow of work assumed in previous sprints was disrupted, which caused unnecessary slowness and also cost overhead. Documentation necessary for the maintenance personnel was to be created before the handover, and as such did not necessarily contain all the required information and details. Mandatory use of ITIL processes when building the production environment was one of the main schedule hindrances of the project according to the interviewees.

4.2.2. Team Structure and Tools

Depending on the items in the current sprint backlog, the team was divided in two or three geographically separated locations during the whole length of the project. The organizational separation of the developers resulted in situation, where even the persons based on the same location did not necessarily sit in the vicinity of each other or communicate with other team members directly. The central location for the project, and the physical location of the server platform was Helsinki, Finland, but the team members were divided on several sites. The Scrum Master performed most of her duties remotely, without being in direct contact with the developers except rarely. As usual in large ICT service companies, almost all developers were also involved in other projects at the same time. The overall experience of the team was deemed very high, although in infrastructure work the use of agile methods is not very common, and is customer dependent at best. As per this fact, most personnel were mostly inexperienced with Scrum, although they received basic Scrum training before and during the project. Use of Scrum was reflected by the use of collaboration and project management tools, most importantly Atlassian JIRA[5] specifically customized for the agency's use. The Scrum Master promoted and demanded the use of JIRA as reflecting the work performed in daily sprints. The Product Owner's most visible role was following the project's progress based on what team members reported on this tool. In general, the team was reported to be happy or at least content with Scrum, at least up until the production environment building phase where ITIL processes broke the team's workflow.

4.2.3. Security Story Types and Their Implementation

The requirements called primarily for well-documented software quality and component and process security. Most of the additional work was directly security related, and creating its documentation. The platform also had strict and formal requirements for availability and reliability. Outside the security domain, the main source of regulation-related work was duplication of all infrastructure into the service provider's second data center. The data centers themselves, as well as the personnel administering the system and its infrastructure were subject to meticulous security screening. Proper level of access control was enforced, the server rooms' CCTV system extended to cover the new servers, and remote connection practices were reviewed. All personnel involved with the client was to be security checked by the national Finnish Security Intelligence Service[6]. Data itself must reside within the country's borders and even the infrastructure's configuration data and work tickets in the Configuration Management Database (CMDB) were to be made inaccessible for personnel who are not security checked.

4.2.4. Technical Tasks: System Hardenings

As an infrastructure project, the main technical obstacle was securing the hardware, operating systems, middleware and the application (the IDM system) against security threats. The bulk of this work was performed by one of the interviewees, the security developer. Hardening in this case covered analyzing and removal, or blocking, of hardware and software features, and testing against the threats. The purpose is to reduce the attack surface of the platform under construction and protect it from both internal and external threats, as well as minimize the components where potential future vulnerabilities may emerge.

On hardware level, hardening means controlling the network interfaces and the surrounding local area network, routing and traffic rules. It also covers any and all hardware maintenance interfaces, typically accessible through the network. On operating system and software level, the operating system's or software manufacturers, such as Microsoft, provide their own hardening instructions which were used as a baseline. These were combined with the best practices of the consultant company's own experiences and policies, and the explicit instructions and requirements given by the client organization. These included uninstalling a large number of modules and services, disabling a number user accounts and policies, and enforcing a number of others, and restricting access and privileges throughout the system. The same principles were applied to each software component installed on the server platform.

By definition, all access rules and user validations had to be applied to the infrastructure services provided for the server platform; these include software and hardware patching, network access, malware protection, hardware and application monitoring, and backups. The inherent uncertainty of security testing, together with the inter-dependency of the components affected by the removal and alteration of the services and restriction of rights made predictable time-boxing of these tasks so unreliable that the team decided to execute them as spikes.

4.3. Cost of Security Work

The Scrum Master estimated that the extra work caused by the regulations was approximately 25 to 50% of the project's total work load. As accurate billing information was not available, this was accepted as the best estimate of the real cost of the security work. Most of the overhead comprises from the documentation of the solutions. Security-related documentation was created by all team members: project manager and the security developer (architect) created most of the documentation, and the Product Owner as the client's representative made sure that the correct regulations were applied.

Developers were burdened by creating appropriate level of security-oriented technical documentation of all their work, especially related to operating system and application hardening procedures. The hardening process itself lasted for four months, presenting the largest tasks in the project. Changes to the production environment were further complicated by ITIL-based requirement of strict Change Advisory Board processing of each change that was made.

5. ANALYSIS AND DISCUSSION

The research objective for this study is to identify best practices as well as hindrances of using agile software development. This case provides a good view how unmodified Scrum lent itself to a situation, where a large amount of regulations caused extra work with uncertainties in work estimates. Due to these uncertainties, or the large amount of presumably indivisible work included in some of these tasks, the team was simply not able to fit certain features into the sprint structure. Also, in contradiction to traditional security view, iterative and incremental approach to development and building forced the project team, steering group and also the client to rethink how the end product's and its management's security assurance was to be provided. In a sequential waterfall model the security deliverables and tasks were tied into the predetermined milestones, without the flexibility provided by Scrum. As presented in Figure 2, the project was in practice executed partly following a `waterfall' model, yet without milestones fixed in advance; these waterfall processes ran alongside the main project, and their deliverables were then included in the project outcomes.

Based on the above, in the strictest sense the project organization failed utilizing Scrum methodology to create the product, although the superficial requirements were fulfilled -- customer was mostly interested in progress reports and the timely delivery of the complete and standard compliant end product. The failures were partly due to inflexibilities on both the company developing the system, and the client demanding a formal and fixed approach to Scrum. Sprint planning for tasks, for example, called for features to be completed during the sprint. When this was already known to be extremely unlikely, these features were agreed to be performed as spikes. In retrospect, this was most likely caused by the thinking that security features were perceived as overhead and not actual features in the product, while in reality the security features were essential to the product itself.

Even without applying any formal modifications to Scrum, at least one of the "secure Scrum" features, presented in Chapter 2.1 and Figure 1, was taken into use, as the project architect assumed the role of security developer. In practice, most of the physical work triggered by security requirements was done in spikes outside the sprints. When the work is done in a non-iterative way, just letting them run along the project, the benefits of Scrum are lost. Based on the project manager's estimate of cost increase factor is 1.5-2x, caused by the security features, and thus there exists a large saving potential in rearranging the security work. Attempting a new approach and restructuring the work into iterations is recommendable in future projects. Initial spikes are acceptable, but in this case the team failed to utilize the experience gained from them, and continued to implement similar security features as spikes even after the first one. This is represented in Figure 2 by the OS hardening spikes H1, H2 and H3. The team defended their selected approach by stressing the inherent differences in the physical environment and management practices of the development, quality assurance and production environments, but also from the undertones of the developer's interview, it was perceivable that the attitude towards using Scrum in this kind of project was negative to start with. Time-boxing the uncertain tasks to three-week sprints, having to perform the demonstrations after each sprint, and other Scrum routines were perceived to some degree as distractions from the main work. This mentality seemed to affect some members of the team despite the personnel was trained in the Scrum method and the tools necessary.

During the interview, the team was quite uniform on the key success factors of the project. They emphasized the importance of document management, and very strict requirement management. The amount of overlapping and sometimes outright conflicting security requirements even within the VAHTI requirements increased the Scrum Master's workload substantially. Use of Scrum was deemed to have overwhelmingly positive effect, by enabling faster reaction to changes in the requirements and directness of the client feedback. Also the team praised the frequent sprint planning's effect of keeping the team focused, in comparison to the very long spikes run during the project. In retrospect, the team regretted not utilizing the Product Owner more already in the beginning, as direct channels to the client were viewed to be very valuable during the implementation. Also, the client's key personnel were not

always present at sprint demos, which caused unnecessary questions and insecurity on the client's side, despite the features were already completed and already once comprehensively demonstrated.

The effect of Scrum to the efficiency of the work was estimated very positive. The extra cost of the security was partly compensated by the fact that rigorous testing and documentation of the technical solutions had also a positive impact on the quality of the work, improving the system's reliability and availability. It can also be argued that the cost of security work is lower when it is done proactively rather than repairing an old system or trying to recover a breached one.

6. CONCLUSION AND FUTURE WORK

This study has presented a case of building an infrastructure and setting up an identity management software platform for a governmental customer. The customer agency had their own set of security regulation and requirements, namely the VAHTI instructions. In addition to the government requirements the service provider contracted to build the system was committed to several international ISO/IEC standards, as well as their own management frameworks and sometimes complex financial reporting rules. Both the agency and the service provider's project management offices required employing the Scrum methodology as the project management framework. The research was conducted in a post-project semi-structural interview, and the information was gathered based on their experiences and notes of the project. The parties involved are anonymized, and only publicly available information about the project and the regulations involved was to be disclosed.

Scrum was initially applied in its standard form, with no formal security extensions. Security engineering activities were integrated into the product backlog, and performed within sprints whenever possible. During the project, the team adapted to the security work by creating a *de facto* security developer role, and many of the security engineering tasks ended to be performed outside of the regular sprint structure: typically, security assurance is based on evidence gained through security testing, which also in this case had an adverse effect on the team's ability to schedule and time-box the items that were subject to these tests; these were performed as spikes instead. The same technique was also applied to documentation, which was performed outside the main sprints, and audits and reviews, which were separately scheduled one-time tasks. The results of these spikes were still presented in sprint demos among the other artifacts and results. The reported issues at product deployment in production environment prompt for developing and applying a delivery model that provides the required security assurance without the interruption to iterative development.

The team viewed the use of Scrum as a positive factor to project cost and quality, although arguably Scrum was not utilized to the maximum extent: important parts of the work were done in spikes outside of the main sprint flow, without attempts to utilize the experience gained from them to time-box the future tasks. This was seen to benefit the project, although an iterative and more exploratory approach to those tasks might have proved more benefits in the long term, and it is still a possibility that the experience gained in this project can be utilized in similar future projects. The project team still regarded the security engineering activities and providing the required security assurance to compose a significant amount of extra work: at final stages, the work load effectively doubled. The initial approach in this project was more or less an unmodified textbook example of the Scrum method, but the team applied naturally certain security extensions. Simply conducting weekly product backlog refinement sessions was deemed essential for the project's success.

This project was a model case of two large entities that have decided to fit their organizations to work according to an agile framework. The nature of work itself has not changed, although the introduction of growing amount of security engineering and increasing regulation put an additional strain on the project's requirement management. Agile methods have inherent preference to produce working solutions instead of spending time documenting them; in contradiction to this goal, the documentation of the solutions is a key deliverable in the field of security. Scrum will continue to be used by both organizations, and as the team's experience grows, we expect also the cost of

the secure systems development to drop, while their quality and security gets better. Based on the experiences gained in this case, Scrum has shown the potential to be suitable for security-oriented development work. With certain additions and modifications, it can be used to provide the security assurance required by the regulators in the ICT and software industry. Especially when applied by an organization capable to adjust itself to fully utilize the flexibility of incremental agile frameworks, instead of partially reverting back to sequential mode of operations. We are yet to observe a pure agile project where security standards are in a central role: truly integrating security engineering processes and security assurance activities without losing the agile values and benefits gained by the use of those methods is still a work in progress.

ACKNOWLEDGMENT

The authors gratefully acknowledge Tekes - the Finnish Funding Agency for Innovation, DIMECC Oy, and the Cyber Trust research program for their support. This paper extends a conference paper "Case Study of Security Development in an Agile Environment: Building Identity Management for a Government Agency" by Rindell, Hyrynsalmi & Leppänen (2016).

REFERENCES

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

Baca, D., & Carlsson, B. (2011). Agile development with security engineering activities.*Proceedings of the 2011 International Conference on Software and Systems Process, ICSSP '11* (pp. 149-158). ACM.

Beznosov, K., & Kruchten, P. (2004). Towards agile security assurance. *Proceedings of the 2004 workshop on New security paradigms NSPW '04* (pp. 47-54).

Creswell, J. W. (2003). *Research Design: Qualitative and Quantitative and Mixed Methods Approaches* (2nd ed.). Thousand Oaks, California: SAGE Publications, Inc.

Diaz, J., Garbajosa, J., & Calvo-Manzano, J. A. (2009). Mapping CMMI Level 2 to Scrum Practices: An Experience Report. In Software Process Improvement, CIS (Vol. 42, pp. 93-104).

Fitzgerald, B., & Stol, K.-J. (2014). Continuous software engineering and beyond: Trends and challenges. *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, RCoSE 2014* (pp. 1-9), New York, NY, USA. ACM. doi:10.1145/2593812.2593813

Fitzgerald, B., Stol, K.-J., O'Sullivan, R., & O'Brien, D. (2013). Scaling agile methods to regulated environments: An industry case study.*Proc. of International Conference on Software Engineering, ICSE '13* (pp. 863-872). doi:10.1109/ICSE.2013.6606635

FMoF. (2009) ICT-toiminnan varautuminen häiriö- ja erityistilanteisiin. Retrieved from https://www.vahtiohje.fi/web/guest/2/2009-ict-toiminnan-varautuminen-hairio-ja-erityistilanteisiin

FMoF. (2012) Requirements for ICT Contingency Planning. Retrieved from https://www.vahtiohje.fi/web/guest/2b/2012-requirements-for-ict-contingency-planning

FMoF. (2012) Teknisen ympäristön tietoturvataso-ohje. Retrieved from https://www.vahtiohje.fi/web/guest/3/2012-teknisen-ymparistön-tietoturvataso-ohje

FMoF. (2013) Sovelluskehityksen tietoturvaohje. Retrieved from https://www.vahtiohje.fi/web/guest/vahti-1/2013-sovelluskehityksen-tietoturvaohje

Ge, X., Paige, R. F., Polack, F., & Brooke, P. (2007). Extreme programming security practices. In Agile Processes in Software Engineering and Extreme Programming, LNCS (Vol. 4536, pp. 226-230). doi:10.1007/978-3-540-73101-6_42

ISO/IEC. (2008). Information Technology - Security Techniques - Systems Security Engineering - Capability Maturity Model (SSE-CMM) ISO/IEC 21817:2008.

ISO/IEC. (2013). Information Technology - Security Techniques - Code of Practice for Information Security Controls. ISO/IEC 27002:2013.

Pietikäinen, P., & Röning, J. (2014). *Handbook of the Secure Agile Software Development Life Cycle*. Univ. of Oulu.

Rindell, K., Hyrynsalmi, S., & Leppänen, V. (2015). A comparison of security assurance support of agile software development methods.*Proceedings of Proceedings of the 15th International Conference on Computer Systems and Technologies* (pp. 61-68). ACM. doi:10.1145/2812428.2812431

Rindell, K., Hyrynsalmi, S., & Leppänen, V. (2015). Securing Scrum for VAHTI.*ProceedingsCEUR Workshop*(pp. 236-250).

VersionOne. (2016). 10th annual state of agile survey. Retrieved from https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf

Yin, R. K. (2003). *Case Study Research: Design and Methods* (3rd ed.). SAGE Publications, Inc.

ENDNOTES

[1] https://www.vahtiohje.fi/web/guest/home_
[2] https://www.viestintavirasto.fi/en/cybersecurity/ficorasinformationsecurityservices/ncsa-fi.html_
[3] http://www.itil.org.uk/_
[4] https://www.vahtiohje.fi/web/guest/708 (available in Finnish only)
[5] https://www.atlassian.com/software/jira/agile_
[6] http://www.supo.fi/security_clearances_

Paper VI

Industry Survey of Secure Engineering Practices in Software Engineering

VI

Kalle Rindell and Johannes Holvitie and Jukka Ruohonen and Sami Hyrynsalmi and Ville Leppänen (2019). *Article in Review*, ( ):

Security in Agile Software Development: a Practitioner Survey

Kalle Rindell[a], Johannes Holvitie[a], Jukka Ruohonen[a], Sami Hyrynsalmi[b], 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*

Abstract

Context: The aim of software security engineering is to define, implement and verify information security for software products. This is achieved by introducing security engineering elements into the software development process, generally by following a software security development life cycle model, or a security maturity model. Agile software development methods, dominant in the software industry, are conflicting with some of the security engineering practices, and the regulatory security requirements.
Objective: To empirically verify the usage and impact of software security engineering and accompanying agile software development practices, currently utilized in industry software development projects.
Method: A survey (N=62) was performed among software and security engineering practitioners, regarding their usage of 40 common security engineering practices in accordance with agile software development activities, processes and artifacts. For the used security practices, the perceived security impact was asked. The results are reported using an analytic framework drawn from extant literature, and research suggestions are provided.
Results: Use of agile software development has an effect on the selection of security engineering practices. The perceived impact of the security practices was lower than the rate of usage would imply, suggesting a selection bias caused by either peculiarities in the security engineering practices, or difficulties in applying the security engineering practices into the iterative software development work flow.
Conclusions: Security engineering in software development mostly conforms with agile practices. Agile usage correlates with usage of security engineering: when the use of activities categorized as security engineering practices increases, the use of agile practices is also more systematic. The use of security engineering is concentrated in implementation phase. Results show a discrepancy between the usage and perceived security impact of the activities, indicating a need for better integration of security engineering methods and tools into the processes and toolchains in software development.

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

1. Introduction

Software security in software development is implemented by utilizing security engineering activities and processes [72, 39, 2]. These security practices are performed in conjunction with the software development processes, presumably by following a security development life cycle or a thoroughly-implemented security maturity model. In agile software development, currently predominant in the software industry, the development processes are not always predetermined, nor are they necessarily predictable at the start of the project. This premise renders prescriptive security models less applicable, and potentially even detrimental to the software development process, setting constraints to the flexibility gained through the use of agile methods and principles [cf. 61, 68, 51].

Security is recognized to be an essential characteristic of "good" functional software. It is also increasingly a regulatory requirement. As a result, security engineering – traditionally thought to be mainly a maintenance concern – is increasingly integrated directly to software development. Software security aims to production of secure software products, with less security flaws and preferably more security-enhancing features. It also concerns the production of development-time security assurance for various stakeholders – customers, regulators, the management, and security auditors. Software security engineering techniques have much in common with software quality improvement efforts, and share many principles with those aiming for software safety.

Security incidents and threats are a significant factor in determining the cost of creating and maintaining a software product, throughout its lifetime [42]. Security of a software product is typically implemented by a progression of security-related activities: setting security objectives, eliciting security requirements, creating a security architecture and design, implementing the security features, and finally verifying the implementation. Rudimentary security assurance consists of various security-related documents, such as security architecture and technical documentation, and, for example, implementation of a security

logging system. Regulation and extensive security requirements may call for additional security assurance. This comes in form of formal security reviews, penetration test and other security testing reports, and security audits [see e.g. 27].

In order to produce software that not only complies with security regulations, but also provides effective security solutions and mitigates the security risks, software security engineering builds the security directly into the software. On the other hand, software development organizations are under strict budgeting and scheduling restraints, resulting in a high demand for increased efficiency and efficacy. To date, it is not known how software development organizations achieve this in general: case reports typically involve rather specific security needs calling for special measures, which in themselves may be interesting, but less useful outside that specific context. This survey was set up to find out how state of the art software development and security engineering methods are applied in the industry. The survey concentrates on the individual activities used in agile software development and software security engineering, extruded by common security development life cycle and maturity models. To provide context for the security engineering activities, also the prevalent agile practices from the most used software development methodologies were surveyed.

This article reports the results of an industry survey, in which the usage of 40 common software security practices was surveyed in combination of 12 agile practices. The research target was to establish how security engineering is implemented in agile software development processes, and to find out how the various security and software engineering practices are interrelated. To gauge the effect of the security practices, also their perceived impact was asked. The survey was conducted among selected employees of 303 software development companies in Finland, in the end of 2017 and in the beginning of 2018. The respondents were software development practitioners proficient with security engineering. This is reflected by the reported requirement for compliance with formal security regulations in 40% of the respondents' projects, and with 67% performing software development under specific security guidelines.

The results reveal the specific security practices in the industry and their security impact. The software industry has adopted the agile practices thoroughly, and security-intensive software development is not an exception: the results implicate that security engineering practices largely conform, and are used in concert, with the agile practices. Agile usage correlates with usage of security engineering: the use of security engineering practices increases the use of agile practices. However, the perceived security impact of these activities does not correlate with their level of use, and especially regulation-driven activities are seen as an inefficient security measure. The software security development life cycle models in general are largely customized, with several suggested security practices being abandoned. In industry context, the selection of security engineering practices appears to be made based on their suitability and compatibility with the prevalent software development practices.

The next chapter provides theoretical background and motivation for the survey. Research methods and the research questions, and the design and structure of the survey, and description how it was conducted are presented in Chapter 3. The survey's results are presented in Chapter 4, and the results are discussed and the conclusions presented in Chapter 5, with practical and research implications, and considerations for future research.

2. Background

2.1. Software security engineering

The purpose of *software security engineering* is to mitigate security threats to the software and data assets [39, 2]. The goal of ensuring availability, integrity and confidentiality for *authorized users* involves a scheme of security policies, which are implemented into software security features during software development, and producing the required security assurance. 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 core issues software security engineering is set to resolve are well-known – and recurring – issues in software implementation and design [66], and typically addressed by "check lists" [45, 55, 26]. Software security engineering provides means to systematically address these security issues in software development.

In formal security management models, security in software is achieved by defining and implementing security controls, enforcing predefined security policies. Policies, in turn, are derived from the organizational, normative and technological context in which the software is developed and, eventually operated. Software security engineering is part of larger security engineering scheme, and is used to ensure the necessary security mechanisms are in place within the software itself, and not as an afterthought, detached from the asset it is supposed to protect [2].

Software dependability is achieved through such means as fault prevention, fault tolerance, fault removal, and fault forecasting [3]. Various software errors, whether caused by accident or malice, are a generic example of dependability threats. From the dependability aspect, security engineering also increases fault tolerance by aiming to prevent errors, both intentional and accidental, and by providing additional mechanisms to recover from errors in a controlled manner. In software development, the errors can be caused by faulty requirements, caused by design flaws, or be introduced in the implementation phase as software bugs. If these latent or dormant security errors are not detected during the development time, these may

later expose software weaknesses that may lead to vulnerabilities, which in turn may be exploited during the actual operation of the software.

Software security engineering is targeted to elements within the software system boundary, and the interfaces between the software and its environment and its users: humans, or other systems. In the software life cycle, it takes place during the software development life cycle phase, containing processes linked to other phases [cf. 1]. In information security, the assumed stance is generally reactive: software security engineering, on the other hand, aims to proactively mitigate the security risk by reducing the threats to a software system's security. Common characteristic of security faults, such as implementation bugs introduced during development, is their persistence, requiring constant maintenance activities. Eliminating such faults already in development requires significantly less resources than later in the life cycle [49]. In contrast, errors in hardware, user errors, and other faults in the operating environment are of a more transient nature and mitigated by means independent of the protected system assets [25].

Systemic *security assurance* is the means to provide proof of the implementation and existence of security. It mainly consists of documentation of the technical security features and security architecture, and the procedural security instructions. In addition, a number of security verification techniques may be used. These techniques, such as various forms of security testing, security reviews and security audits, seek to detect these flaws and errors, and to accumulate the required level of assurance. Security assurance requirements may be normative or organizational, form the proof of existence and functionality of the security controls used to enforce security policies. In the technical context, security assurance can also be utilized e.g. for security metrics and monitoring, as well as for incident detection and security forensics.

Security assurance has a significant role in security regulation. Historically, the primary goal of software security, in the context of software development, has been to provide the regulator or the procurer – such as a bank, a branch of government, or a military organization – sufficient proof of security [76, 77]. In addition to the reviews and verification, the security policies at higher security levels may include security audits, performed by external auditors, regarding the processes, tools and personnel involved in software development. The security proof is not restricted only to the technical software, but also to processes and personnel for protection against "malicious insiders". The Secure Software Engineering Capability Maturity Model (SSE-CMM) [29], an ISO/IEC standard, claims to contain the best practices in security engineering, aiming to formalize the security work into an exhaustive set of security processes. This standard defines 129 security processes in 22 process areas, 11 of which directly concern software development activities. The other 11 process areas are organizational and management processes.

To make the investment in software security more feasible for software development organizations, other less demanding maturity models have been developed. An example of such is the openly available Software Assurance Maturity Model (SAMM) [46], claiming to be "agile agnostic". Complimented with a diverse set of open-source tools also provided by the Open Web Application Security Project (OWASP), the SAMM offers a security framework for organizations seeking to improve their security, but not formally required to comply with SSE-CMM's maturity levels. Developers and organizations are encouraged to tailor their own processes and perform only the necessary security improvements. The design and implementation of secure software is guided by industry best practices, published in the form of security checklists, and "don't do"-lists [45, 55, 26].

A multitude of national and domain-specific security regulation guides the security work [21, 49]. In Finland, where the survey was performed, a comprehensive set of information security instructions has been issued by the Government Information Security Management Board (Finnish abbreviation: VAHTI [69]). These instructions are *de facto* regulatory security requirements when working with the government information systems. The instructions include a three-level maturity model for security in software development [70]. This specification also contains a Software Security Life Cycle (SSDLC) model, and was used as one of the sources for security practices in the design of this survey.

Created explicitly to improve the security of software products, SSDLC models offer concrete security activities directly in contact with the software development processes. This model of security improvement, significantly simplified from the maturity models, stems from the work performed at Microsoft to improve the security of the Windows operating system. This led to the publication of Microsoft SDL [72]. The life cycle approach was further refined, resulting in the Touchpoints for Software Security, with elaborate discussion and instructions for applying them in a software development organization [39]. This model also forms the development life cycle model included in the Building Security In Maturity Model (BSIMM). Microsoft later made an effort to adopt the SDL for agile development, although in a very rudimentary manner [24, 40]. The SAMM and also SSE-CMM contain elements that can be used to constitute an SSDLC. In ISO/IEC standardization, software security is formalized in the Common Criteria [28], which also includes elements of an SSDLC. All of these were sources for the VAHTI specification, and were used as sources from which the activities were selected for this survey. These security life cycle models are the main source for security engineering practices in this study.

2.2. Related work

In software development, agile methodologies promote and provide the ideals of flexibility and freedom [7, 58], an approach inherently different to the ones promoted by

the maturity models, predating the agile methods. Even though the agile ideals were initially criticized due to their perceived contradictions with the older process-oriented approaches [67, 43], agile principles have been successfully adopted also to the regulated areas of the software industry, with extensive safety and security requirements. The iterative methods also contain inherent mechanisms for continual performance and quality improvement, and agile principles build upon – and simplify – that tradition [see 60]. Agile development has enabled new continuous delivery models, such as DevOps, which considerably shorten the maintenance cycles and, in security work, the delivery of security improvements. However, in regulated environments, DevOps has also some definite challenges, reflecting the challenges of introducing agile practices and organizations [34]. Outside the case studies of singular organizations or projects, very little empiric research exists on how security engineering has been adapted to the agile software development, or vice versa.

As agile methods gained popularity, applying security engineering methods and models into agile software development was seen challenging [73, 15]. Even recent empiric research indicates troubles in the adoption of agile methods in security-intensive contexts, indicating organizational or technical difficulties [68]. The ideals of a lightweight organization, non-deterministic execution of development actions, and the unpredictable trajectory of development and related processes, create a contradiction with the basic premises of the more deterministic security development models. As an offshoot of the security engineering tradition, even many of the security practises in the SSDLCs are inherently difficult to integrate into agile work flow. Furthermore, security regulation may impose requirements also for the development process, such as formal process audits, or compel the use of a maturity model either formally or semi-formally. A fundamental mismatch between formal security engineering and software development has promoted many attempts to combine CMM with agile processes [59, 61]. Agile processes can even be beneficial for reaching formal maturity levels, with examples of even CMM level 5 – the highest – adapted, while still retaining at least some agile characteristics in the software development process. However, applying a high-level CMM in an agile organization is characterized by heavy customization of the typical agile processes: the goal is to augment the processes with various "non-agile" elements while, still attempting to preserve the core agile principles and values. This combination is not easy to achieve: rigid and formal processes and protocols will cause overhead and organizational micromanagement, significantly reducing the velocity of development work.

The challenges being well known, aspects of software security engineering in the agile context have been examined in a body of previous work. Early contributions for agile software security engineering were mainly theoretical concepts how to perform and manage agile software security engineering. Examples of such research were suggestions for security architectures in agile development Chivers et al. [13], Ge et al. [18], as architecture work is deemed one of agile methods' weak points [9, 60]. Similarly, suggestions for producing security assurance using Extreme Programming (XP) were made by Beznosov and Kruchten [11], also providing outlines for providing assurance at various phases of security development life cycle: requirement, design, and implementation assurance, including the use of security testing. Use of misuse cases, one of the security engineering activities included in this survey, was reported by Kongsli [33]. Tondel et al. [65] performed a review of security requirement elicitation. In support of continuous delivery, Ben-Othmane et al. [10] focused on finding and acceptable a level of *continuous software security* using agile methods.

A major research challenge centers around the issue of how to integrate the security engineering practises, maturity models, and activities required to comply with related standards into the flexible agile work flow [50]. The main objective is not to maintain 'agility' as such, but to produce as secure software as necessary, as efficiently as possible. The quality-improving mechanisms in agile development – iterative work, retrospectives, constant refactoring and continuous integration – work towards both of these goals when producing secure software[6]. Also the software security life cycles in agile context have been a frequent subject in software security research [4, 12, 5]. General challenges in adaptation of security engineering into agile development have been identified [20], prompting further research in the subject. An effort regarding the utilization of software security engineering was focusing on the relationship between security engineering usage and security education [47].

Also some related surveys have been performed. Survey by Licorish et al. [35] provided information about the relationship of technical debt management and agile practices. Producing secure software requires high software quality, and the technical debt management and other quality improvement techniques share many characteristics with security improvement. Notably, the agile software development practices selected to that survey are directly comparable to the ones in this security survey. In Finland, a survey concentrating on agile and lean usage was conducted by Rodríguez et al. [54], focusing on agile method usage. Both of these surveys had a specific focus on agile software engineering, although neither of these studies touched the subject of software security. In another related survey in Finland, Savola [56] concentrates on information security skills; this governmental survey describes the security education background and skill gaps in information security work. In that study, the overall security skill level was deemed adequate, but lack of work force sufficiently skilled in security was reported a problem by 60% of the companies, prompting an increase in security education. Training and security awareness has been found to have a direct positive effect to the use of security practises [42].

The Building Security In Maturity Model (BSIMM)

contains an annual industry survey (for the last survey see [63]). Reporting in BSIMM is limited to usage frequencies, and the respondents are mostly in executive position. The reported fact that all the companies participating in the BSIMM survey have an established software security group provides a homogeneous set of respondents, possibly suggesting a sampling frame bias. Focusing on companies with notable security investments may also cause bias in selection of security practices. These include excessive usage of activities required to pass formal security audits not only for the software product, but also for the development organization – a rare requirement even in security-intensive software development. BSIMM also makes a claim not to be prescriptive, yet it contains clear recommendations for software security engineering in the form of *12 core activities "everybody" does*. These 12 activities were among the 40 software security engineering activities selected for this survey, with distinctly different reported usage compared to the BSIMM results [53].

Apart from a small web survey (N=46) concerning the general adoption of security life cycle usage conducted in 2010 [19], no surveys about industry use of the security engineering practices in software development are known. The survey reported in this article fills this gap in research.

3. Research design

Research described herein executes a survey on a target population of software engineers, software security specialists and other direct participants of the software development processes. The survey measures and correlates perceived use and impact of established security engineering activities and agile software engineering activities. Targeted population is characterized via sections of the survey prompting respondents' personal (experience, education, role, amount of applicable software development projects, and specific security training acquired) and organizational (company size, application area and possible regulations applied in projects-under-delivery) backgrounds [see 75, 48].

On a higher abstraction level, in addition to activities taken during the software development process, the respondents were asked to name the agile methodology they have used, if any, and also any security regulation or guideline they have been following. This provided contrast, and further reassurance, for the queried activity-specific answers. For ethical reasons, the respondents could opt out from answering the background questions.

3.1. Research questions

The objective of the authors' research undertaking was to understand the relationship between software engineering and security engineering practices, and their reciprocal impact. Three research questions were specified to reach the objective:

RQ1: How is security engineering implemented in software development?

Security in software development consists of security engineering activities. These activities work towards improving the security of the software product by directly addressing security-related issues (e.g. vulnerabilities) and enforcing security policies. Security policies include the direct security mechanisms, such as authentication and encryption, and the means to provide security assurance, such as logging and various types of security documentation.

Hence, security engineering in software development must be implemented by executing the specific security engineering activities at suitable parts of the software development life-cycle. A baseline for security engineering in contemporary software development is pursued by posing RQ1.

The question is addressed by selecting 40 software development security practices from ISO/IEC Common Criteria, Microsoft SDL, BSIMM and VAHTI. The practices are divided into five groups which represent the various phases in the selected security development life cycle models. For each group, a set of questions in the survey query how systematically the practices are used.

RQ2: What is the perceived impact of software security activities with respect to their (level of systematic) use?

Having established how security engineering is executed as part of contemporary software development in RQ1, we must establish if the execution actually meets the goal of increasing security. Hence, RQ2 prompts the perceived impact for the exercised software security activities. Further, the question must also be sensitive to the level of the security activities' systematic use, as this impacts attained level of security. To further motivate RQ2, software security consists of several types of activities, but a software project typically cannot incorporate all of them. Finding the most effective security activities is crucial to producing secure software in a cost effective manner.

This question is addressed by further prompting the perceived security impact on top of the level of systematic use for the pre-established 40 software development security practices. The survey is constructed in a manner which allows the respondents to only be prompted the impact for those activities they have previously indicated some level of use as part of their contemporary work.

RQ3: Does usage of agile practices and security engineering activities correlate?

As previously discussed (see Section 1) the iterative and incremental agile software development is an ill-match to many security engineering models that require a level of up-front design; possibly diminishing agility. However, agile software development is deemed an effective way to produce software even in demanding environments, it enjoys a

high global adoption rate according to latest surveys, and as such agile development projects constantly have to respond to heightened security demands. As RQ1 and RQ2 establish the state of security engineering in contemporary software development, RQ3 adds to this by exploring possible correlations between reported level of systematic use of agile software development activities and the security engineering practices.

Further, finding dependencies or correspondences between software development and security engineering practices allows the design of efficient and flexible processes to develop secure software; adding to RQ2's efficiency considerations. Addressing RQ3 also clarifies any differences between software development processes with and without security elements. This is an important catering as software development processes are quite well-researched whereas research on software security development practice remains scarce.

This question is addressed by directly asking the respondents for the perceived level of systematic use and development effect for each agile activity, and comparing the attained result with activities found in the analysis of RQ2.

3.2. Survey design

The survey was designed to provide directly quantifiable data on the research questions [31]. All questions related to the research questions were answered on a 5-step Likert scale. In certain questions there was also an option to elaborate on the answers using a free text field. To make the answers more commensurable, the respondents' professional background, some company details, and the application area were also inquired. To broaden the knowledge gained through the survey, also some company details and existence of any security certifications were asked. To gain the necessary knowledge about the agile and security practices, the respondents were asked to concentrate on the last project containing both of these elements.

The design of the survey is presented in Figure 1.

The survey consisted of three main sections. To maintain objectivity in a quantitative survey, some information was gathered about the respondents (overall experience, role, experience with agile security development projects). Organizational information consisted of organization size, the security certifications obtained by the organization, and the application area or the respondent's latest project with security constraints. The main part of the survey consisted two questions about agile practices, processes and artifacts, and of security engineering activities divided into five security development life cycle phases. For each security engineering activity the respondent had used, also their opinion about the effectiveness of the activity was asked.

The design was considered sufficiently accurate from the technical point of view, with anticipation of a potentially detrimental effect to response rate, caused by its length and specific focus on software security. These concerns were tested with a 20-person pilot group, resulting in one explicit concern about the length of the study. Based on the feedback, some corrections to the wording of the questions were made. The structure and overall design of the survey remained unchanged. The answers of the industry practitioners belonging to the pilot group were considered fit to be included in the final results. Before submitting the invitations, the survey design was assessed from the viewpoints of its validity, reliability, applicability, consistency and neutrality [see 62].

3.3. Scales

The agile and security engineering practices were surveyed using the following five-fold 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. Activity was not used

This scale was selected based on a premise that the development process is not fixed in agile software development. Therefore, the value five is phrased with the word "systematically' rather than with the word 'always'. Otherwise the scale was designed to be comparable to the ones used in existing surveys (notably Synopsys Software Integrity Group [63]). The same scale was used in the questionnaire for all security engineering activities.

The questions regarding the impact of security engineering activities were designed to provide data on the activities the respondent had used. As the questionnaire was conducted online, the design allowed certain dynamics to be introduced. To keep the questionnaire as short as possible and to save the respondents from unnecessary repetition, these questions were hidden by default. As the respondent reported activity usage above 1 (Activity was not used), the question for the impact of this activity was made available. The scale for these questions was a slightly altered 5-point Likert scale:

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

The options for evaluation were chosen based on the selection of the sample. The respondents were considered to be practicing software professionals, and only a small proportion of the 40 security practices would be used by a given respondent. The answering process was streamlined by hiding the unused activities, and simplifying the answering process by limiting the options to only positive values. The selection of security practices was based on established industry practices, not a hypothetical framework. The underlying assumption was that the

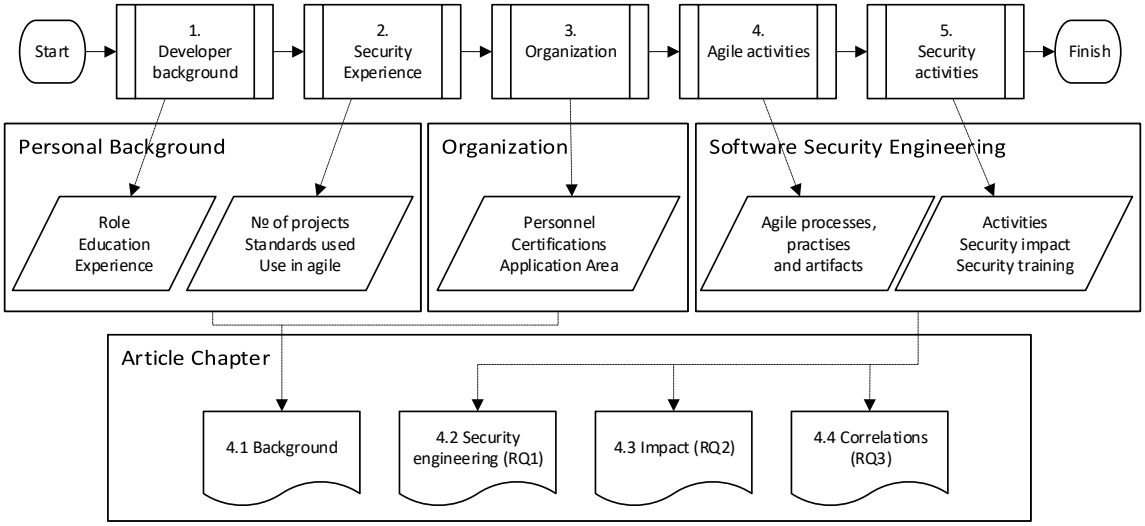


Figure 1: Survey design flow chart with connected research objectives.

non-effective practices, or the practices having an adverse affect to security, would not be used in the industry context.

3.4. Selection of processes, activities, and artifacts

The agile activities, processes and artifacts are the basic elements from Extreme Programming (XP) and Scrum. These were further augmented with more general organizational work practices and processes that are typical to agile software development. Many of these stem directly from the classical Agile Manifesto [8]. More importantly, the sixteen agile work practices are exactly the same as the ones used in a recent study on technical debt [23], sans retrospectives that were omitted. Examples include test-driven development, refactoring, continuous integration, iteration backlog, and rest of the usual suspects.

Security activities were selected from four main sources: (a) the Microsoft SDL model (12 activities), (b) the BSIMM surveys (12 activities), (c) the Common Criteria (ISO/IEC 15408-1:2009), and (d) the Finnish governmental security framework VAHTI. In addition, three activities were identified from previous research [see 51]. The distribution of these activities is presented in Table 1.

Table 1: Security practices in each SSDLC phase

Phase	Practices
Requirement	7
Design	7
Implementation	8
Verification	9
Releases	7

There is a considerable overlap in the practices in the life cycle models used as source. The Common Criteria is the oldest source in the selection. VAHTI, the most recent work, consists only of activities defined in the earlier models, and does introduce any of its own. Security practices selected from VAHTI represent the "basic" level, the lowest acceptable by the standards of the Finnish government. VAHTI remains a recommendation, not an audited standard; the auditing is performed by applying the national Information Security Audit Tool for Authorities, KATAKRI 2015 [30].

As the context of the survey were Finnish software practitioners, the naming of the security activities was, in cases when modificated, following the VAHTI nomenclature. Not all of the associated terms are rigorously defined even in existing research [64]. Nevertheless, the descriptive names for the activities should be understandable to the audience targeted in the survey. To further increase the familiarity among the respondents, the descriptive names were formulated to conform with the Microsoft's SDL and the annual BSIMM surveys.

3.5. Survey implementation

The survey was conducted as an invitation-based online survey. There were two types of invitation: a public web link and targeted links for personal e-mail invitations. The invitees were collected by three methods:

1. A public invitation to the survey was sent to a mailing list containing contact persons of 340 Finnish software companies.
2. A subset of this list containing the web addresses of 303 of these companies was composed; the companies'

web pages were then accessed and further public e-mail addresses were extracted from the contents of these pages.
3. A further subset of 69 companies from the list of 303 companies were targeted by searching the corresponding employees from LinkedIn, resulting in total of 1,436 targeted e-mail invitations.

This multi-stage approach was selected for several reasons. First, the initial public invitation was sent as a part of the Finnish Software Entrepreneurs Association's monthly news letter. This news letter was targeted mainly to the Chief Executive and Information Officers of software companies, who were asked to forward the questionnaire to their employees – not considered the best approach to reach the targeted audience of security and software specialists, but considered important to promote the topic and gain acceptance for the potential use of work time to answer the question. At the time, the association had approximately 340 member companies receiving the newsletter. Additionally, the newsletter was sent during holiday season following a premeditated schedule which could not be affected. The public invitations (method 1) were considered potentially ineffective, and the manual invitations were sent to targeted persons (method 2). The manual process was further refined by involving the use of LinkedIn search (method 3). The idea was to spread the questionnaire to as many persons in the targeted audience as possible. The targeted invitations were sent within a time frame of two weeks to ensure uniformity of the answers. This two-week period was added to the time the survey was kept open.

A similar use case of social media was described by Middleton [41]. The people search in LinkedIn service was performed using the company name as their current employer. The e-mail addresses were composed in three principal steps:

1. The company's website explicitly stated their e-mail addresses were in the form 'name@domain' or 'firstname.lastname@domain'.
2. The search result contained the person's full name.
3. The person's job title did not clearly indicate they do not work in the field of software projects: this excluded non-technical executives, finance, human relations, and sales and support personnel.

Using methods 2 and 3, the survey link was sent to 1,436 e-mail addresses. The survey service did not provide reports of failed deliveries, so the actual coverage cannot be reliably known.

The survey remained open for a period of six weeks, with a reminder message sent to the targeted respondents after three weeks. A public reminder was not sent, as the timing of the next monthly newsletter was deemed too late. When a targeted questionnaire was answered, the targeted link closed by the survey service and could not be reused, a mechanism promoting the reliability of the results.

3.6. Response rate

The survey link was opened 546 times, and 114 responses were given giving a response rate of 21% of this total. Altogether, 63 respondents finished the survey; 48 of these answered after the initial call, and 15 after a reminder sent to those not yet responded three weeks after the initial round. The survey was open for a 6-week period from December 2017 to January 2018; an extended period was seen necessary to compensate for the holiday season, as the invitations were sent to the respondent's work e-mail addresses, and potentially not reached their targets for several weeks. This tactic was validated by the increase of 31% in responses after the post-holiday season reminder message. One single case of vandalization was removed from the result set, producing a total of 62 viable responses with all questions answered.

4. Results

In this chapter, the results from the survey are presented against the research questions framed in Chapter 3. First subchapter presents the respondents' background and organization, and the following three provide an analysis of the results, based on the three research questions respectively.

4.1. Background

4.1.1. Experience, education, and organizations

The respondents are generally well-educated and highly experienced in software engineering. Only about five percent reported no experience in software development – and 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 been trained at the highest doctoral level.

The respondents work in diverse organizations. About 39% worked in organizations with less than fifty employees and about 35% in organizations with more than 250 employees. 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. According to the breakdown in Table 2, this targeting succeeded well: software developers account for about forty percent of all respondents. By also taking the relatively high educational levels and long work experiences into account, the sample can be considered well-representing for empirically evaluating the three research questions – after

all, the security engineering activities surveyed are fairly technical in their nature.

Table 2: Project roles

Role	Share (%)
Developer	40
Architect	22
Project manager	10
Executive	8
No projects	5
Security specialist	3
Product owner	2
Team leader	1

Table 3: 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 queried 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 [71]. The results summarized in Table 3 show no surprises: Scrum and its variants are currently the most popular agile methodologies in Finland. In contrast to other surveys [35, 71], none of the respondents reported working with XP or some variants of it.

While these details about agile methodologies are interesting on their own rights, the bottom line is that only a negligible amount of the respondents appear to be working with some non-agile software development methodologies. Insofar as the scope is restricted to Finland, it is safe to say that agile software development is almost universally used today. The point is particularly important for the survey at hand: there exists no widely used polar opposite for empirical comparisons. In practice, software security engineering is mostly carried out in agile software development settings just like software development in general.

But while practically all of the respondents are 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 practices on a similar five-fold Likert scale used for the security activities (see Section 3.4). When compared to an earlier survey [35], the adoption rates of these activities appear almost consistently higher. Inherently agile items are 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 – are also very popular. On the side of less frequently used practices are daily meetings, collective code ownership, 40-hour week, simple design, pair programming, on-site customer, and test-driven development. That said, even these less frequent activities are still used in the majority of the respondents' projects.

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. According to Table 4, this pillar manifests itself also in the typical security constraints imposed upon the software projects surveyed. In other words, many of the projects have constraints that are context-specific and related to customers' particular security requirements. While another survey would be required to evaluate whether customers are using formal standards for their requirements, it seems that the use of formal standards is generally limited at least on the development side. A large amount of the security constraints are also defined by the software development organizations themselves.

Table 4: 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
Others	6
ISO 15408 (Common Criteria)	4
RFC 2196 (Site Security Handbook)	4
None	2

These are important points because a fundamental problem with formal security standards, such as the Common Criteria, is the omission of business processes that should arguably drive the whole security engineering [2]. In this sense, agile practices can patch the limitations of security standards and checklist-style guidelines. Nevertheless, also standards and regulative frameworks are used albeit less frequently than self-designed security constraints. It is also important to emphasize that no single standard or framework is used exclusively, and national frameworks are slightly more common compared to international ones. The point about national frameworks is particularly noteworthy because much of the existing research has focused on international standards and guidelines used by large multinational companies. Another point is that national frameworks and regulations presumably continue to further intensify the already pronounced proliferation trend impairing security standards, frameworks, and guidelines.

Finally, the survey questionnaire contained also a specific question about whether the respondents' organizations had at some point obtained at least one security cer-

tificate through a formal auditing process. Only 18% of the respondents answered positively to this question. While this observation is partially explained by the sample's imbalance toward web development and cloud applications, it also tells about the generally limited use of formal auditing in the current software industry. In fact, the eighteen percents can be actually interpreted as a quite large value.

4.2. Security engineering in software development (RQ1)

The primary research question about the use of security engineering activities is best answered by framing the activities with the five distinct life cycle phases: requirements, design, implementation, verification, and release phases. The following presentation proceeds accordingly.

4.2.1. Requirements phase

Security requirements engineering is essential to security engineering. The use of activities related to requirements is also among the highest in this survey. According to the results summarized in Fig. 2, the most frequently used activities in this phase are concentrated on the business value side of the requirements elicitation process.

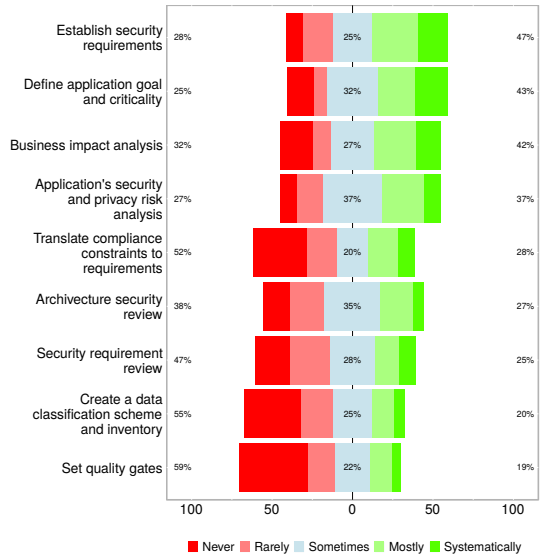


Figure 2: Security engineering activities at requirements phase

The most used security engineering techniques during this phase are simple processes: eliciting security requirements, defining the goal and criticality, and performing business and risk analyses. All are fundamental elements for software security. These activities also produce results that translate directly into the efficiency of software development and the economic value of the software produced. Of the security artifacts, security architectures appear to be only sometimes produced and reviewed. This observation is a small surprise because security architectures are specifically emphasized in the Common Criteria's security assurance requirements. Accordingly, security architecture should be the first strictly security-related assurance document to be produced during software development.

There are also a few other surprises. 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. 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" [63]. The explanation for this diverging result may be simple, however. According to the free-form comments about the survey's design, some respondents commented that data classification schemes are nowadays typically created through use cases. Although use cases may be problematic in the security and safety contexts [22], these are still a highly typical characteristic of agile software development and may thus explain the results. The prevalence of agile development may also explain the limited use of quality gates – an activity suggested by the Microsoft SDL. In a sense, these can be seen as antitheses of agile software development; the enforcement of additional policies may constrain the ideals of flexibility and freedom endorsed in agile development. Furthermore, it is noteworthy that compliance constraints are not widely translated into requirements. A potential explanation may relate to the relatively infrequent use of formal standards and regulations (see Table 4). Because self-designed constraints are commonly used by the respondents sampled, it may be that many of the common compliance questions have been translated into unified requirements shared in all projects of a given development organization.

4.2.2. Design phase

The design phase refers to the work required to translate requirements and abstract architectures into plans for concrete software features and functionality. In other words, the technical context of the software produced is clarified and aligned with the results from the requirements elicitation phase. This alignment involves the updating of the risk analysis with technical risk definitions and mitigation plans. The general software architecture is typically also designed during this phase. By implication, the design phase is often seen as particularly crucial for security engineering [21, 49]. In terms of concrete security engineering activities, this phase involves threat modeling and attack surface recognition, among the other related tasks. When agile software development is used, many of these activities are carried out using security stories and misuse cases [37, 38, 57]. 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 design phase activities are depicted in Fig. 3.

Application security configurations, design requirements, and guidelines for the software architecture and application development are the most frequently used se-

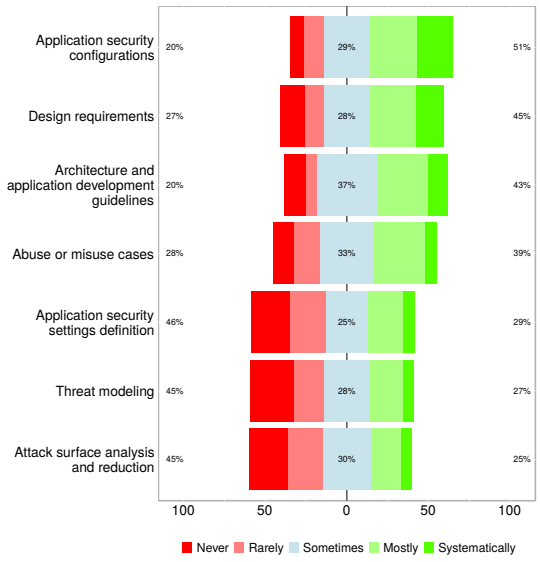


Figure 3: Security engineering activities at design phase

curity design activities. Also abuse and misuse cases are rather frequently used. As these are similar to activities performed in most agile software development projects in general, it seems that adding a security flavor to these common activities is not difficult. In contrast, security settings, threat modeling, and attack surface analysis are explicitly related to security engineering. These are also rather infrequently used according to the results. For instance, almost one third of the respondents had never used threat modeling – even though it is promoted by practically all development life cycle models. There may be also a certain causal logic behind the results. When threat modeling is not done, attack surfaces are not known, for instance. A further examination also reveals that threat modeling was seldom used by those respondents who rarely defined security configurations.

4.2.3. Implementation phase

When an agile software development project enters into the implementation phase, the developers working with the project pick up allocated tasks from the project's iteration backlog. 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 environment (IDE) applications. Furthermore, the tool sets currently used, such as ones suggested by NIST [44], include various effective means to improve software quality, including review functionality and tools for static analysis [14, 74]. The security engineering tasks during implementation reflect these activities and tools thereto. In addition, security documentation, coding standards, and many related activities are typically present in the implementation phase. The results regarding the eight surveyed implementation phase activities are illustrated in Fig. 4.

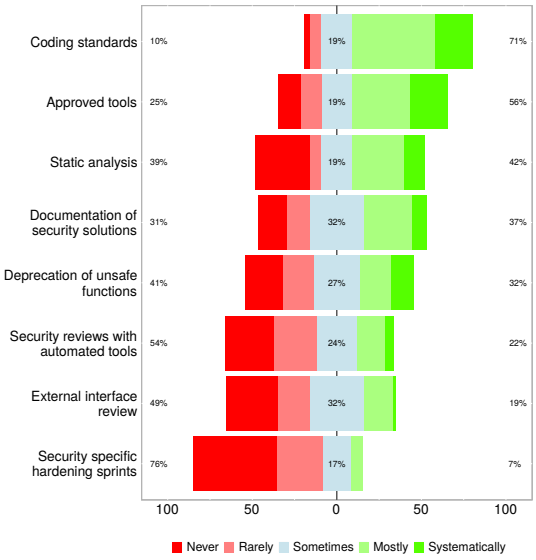


Figure 4: Security engineering activities at implementation phase

Coding standards are frequently used among the respondents surveyed. This observation is not surprising because these are also a part of the common agile principles. The tools used for development are also frequently agreed upon alongside security documentation and static analysis. These observations seem logical in the sense that coding standards, common tools, and static analysis are frequently used irrespective of whether the context is explicitly related to security. Therefore, the more noteworthy results again relate to the activities that are only seldom used. In particular security reviews with automated tools, reviews of external interfaces, and security specific hardening sprints are only infrequently used. Regulation aspects may again explain these results. For instance, security hardening sprints have been promoted in the safety context as a way to ensure regulatory compliance [17]. However, for agile projects dealing with less strict compliance requirements, 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. In other words, these hardening sprints may introduce a certain anti-agile "security gate" into otherwise flexible development work. After all: if the members of an agile project have already built security in, why would they do it over again? However, as will be soon discussed, the answer to this question is not straightforward.

4.2.4. Verification phase

At the security validation and verification phase, software security is verified by testing it against software weaknesses that may lead to software vulnerabilities. The implementation of the security requirements specified is validated by this process. In iterative software development, the potentially production-ready release candidates are further selected as the result of the verification and validation processes. The surveyed activities behind these processes are shown in Fig. 5.

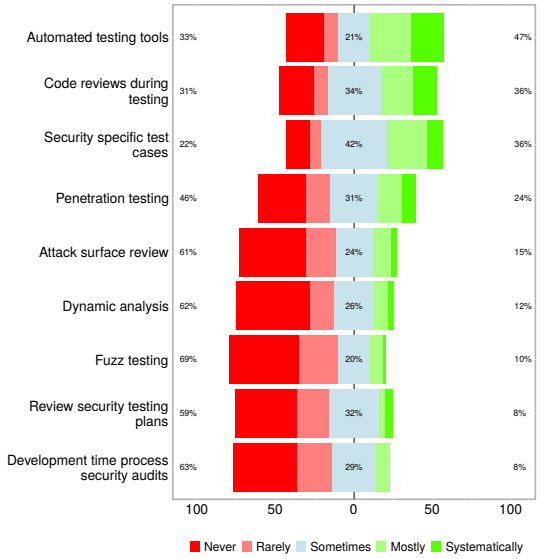


Figure 5: Security engineering activities at verification phase

The results are 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. 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 [22]. 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. Finally, it is 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.

4.2.5. Release phase

The release phase retains many of its traditional functions in a security development life cycle. Even though the extensive use of continuous integration and continuous delivery practises have somewhat blurred the boundaries, many questions related to maintenance must be still addressed. Regulatory requirements also constrain the trend toward full automation of release engineering [34]. Thus, in practice, 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 regarding the surveyed release-time activities are shown in Fig. 6.

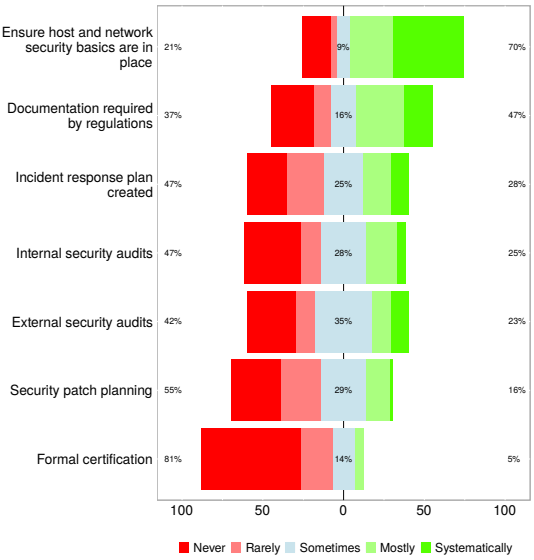


Figure 6: Security engineering activities at release phase

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 [78]. Host and network security essentials can be further seen as prerequisites for other security requirements [53]. The respondents also commonly produce documents required by regulations. In a sense, this observation indicates that security engineering practices can augment the typically minimal documentation produced in agile development. When combined with the relatively frequent documentation activities during the earlier development phases, the results presented do not seem to support the conclusion that agile software development would exclusively rely on

"ad hoc, inaccurate, incomplete, or non-existing documentation" [16] at least in security matters. On the side of rarely used release-time activities, different audits and certificates have been 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.

4.3. Perceived impact versus use of security engineering activities (RQ2)

The survey design provides means to compare the use of the activities in each of the five distinct development phases with their perceived impact upon software security. To provide a concise but sufficient overview, the Likert scales (see Section 3.3) can be used by subtracting the average numerical values between the use and perceived impact of each activity. In other words: for a given activity, a positive value means that on average the activity was used more frequently compared to its perceived impact. A negative value in turn tells that the average use of an activity was "less" than its perceived impact might instruct. The results from the subtractions are visualized in Fig. 7. Although the figure should be interpreted only tentatively due to the rather mechanical comparison, four general but important points are still warranted.

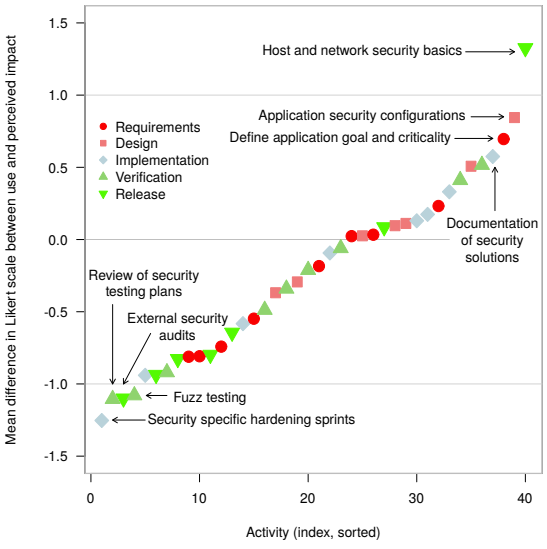


Figure 7: Use and perceived impact of security engineering activities

First, the activities are very poorly aligned in terms of their use and perceived impact. In the ideal case most of the activities would be located around the value zero on the y-axis. As can be seen, only a very few activities satisfy this coarse numerical optimum. Some of the subtractions even yield values outside of the $[-1, 1]$ range. Given the five-fold Likert scales used for the subtractions, such values indicate a particularly large mismatch on average. Second, however, the amounts of negative and positive values are roughly comparable. Therefore, it cannot be concluded that the use of security engineering activities would be systematically subordinating to their perceived impact. Third, there is no apparent pattern in terms of the five distinct development phases. No particular phase is especially misaligned; the general mismatch applies to some activities in all software development phases. Last, there are indeed a few activities that seem to be particularly misaligned. While agile practices may explain a portion of these particular misalignments, another portion may be explained by time and resource constraints as well as the type of software typically produced by the respondents.

Consider fuzz testing as an example: building and configuring an efficient fuzzing framework and toolchain requires a substantial amount of time. In research, as in practice, a typical fuzz testing period is 24 hours for each tested software instance [32]. As fuzz testers keep finding software crashes also after that, a fuzz test suite should be applied for an extended period of time – from several days up to one week. The required effort and the time restraints tend to conflict with a typical agile work flow. Such a conflict may also explain the misalignment reported: even though the respondents perceive fuzzing as an efficient way to improve software security, they also acknowledge the limited use of this testing technique.

An analogous explanation may apply to security related hardening sprints. As was briefly noted in Section 4.2.3, these may be difficult to apply without breaking the agile work practices – even though the potential impact of such sprints is acknowledged by the respondents. Another example would be host and network security for which time and resources are frequently allocated, although the actual impact of such allocations seems limited according to the respondents. Given that most of the respondents are also dealing with web applications and cloud platforms (see Section 4.1.1), it may be that the requirements for host and network security can be replicated across multiple projects. Another explanation may be that traditional security aspects are over-emphasized at the expense of software security.

Whatever the actual explanations may be, the results are sufficient to underline a general mismatch that presumably originates from different context-specific factors. Agility is one but not the only factor. Given that a mismatch is evident already with the forty activities surveyed in this paper, questions related to context-specific prioritization are apparent with more comprehensive frameworks such as BSIMM. There is no one-size-fits-all solution to software security. Nor do the solutions with *perceived* impact necessarily equate to *actual* software security.

4.4. Correlations between agile practices and security engineering activities (RQ3)

The survey questionnaire does not contain explicit questions on how the respondents perceive or understand the linkages and challenges between agile software development and security engineering activities. An indirect statistical analysis is still possible, however. By following existing research [23], a concise exploratory overview can be presented by correlating the common agile practices and processes (see Sections 3.4 and 4.1.2) against the use of the security engineering activities. To examine such correlations, some linear combinations are required due to the large number of variables; in total, there are 16 variables measuring agile practices and processes, and as many as 40 measuring the use of security engineering activities.

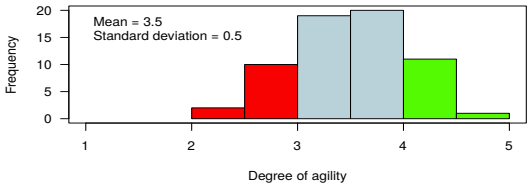


Figure 8: Degree of agility

A sensible approach is to combine the sixteen agile variables into a unified "sum variable" by using the arithmetic mean of the individual variables. There are three reasons why such a combination is reasonable. First, the combined variable is statistically highly sensible according to Cronbach's classical measure for internal consistency ($\alpha = 0.79$). Second, the combined variable is easy to interpret. By reversing the scale such that higher values indicate more systematic use of agile practices and processes, the combination provides a straightforward measure for an organization's "degree of agility". Third, combining the agile practices and processes is contextually reasonable. As can be seen from Fig. 8, all of the organizations surveyed are agile, but some are more agile than others. By further reversing the scales of the security engineering variables, the correlations between these and the organizational degree of agility are summarized in Fig. 9.

All but two of the coefficients shown are positive – it should be also remarked that only one coefficient has a negative sign in an auxiliary ordinary least squares regression. Increasing degree of agility tends to thus slightly increase the use of security engineering activities. Eleven of the Kendall's τ-coefficients are statistically significant at the conventional $p < 0.05$ threshold (see Table 5). Among these is the rank correlation coefficient for security specific hardening sprints. To augment the earlier discussion, it should be therefore noted that such sprints seem to be used in those particular organizations that are especially agile in their software development. That is: if there are adoption problems, these cannot be generalized to all software development organizations. Increasing organizational agility tends to also slightly increase the use of generally unpopular activities such as external interface reviews and development time process security audits. These are generally interesting 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 [52]. In contrast, the results indicate that organizational rigidity tends to decrease the use of the activities surveyed.

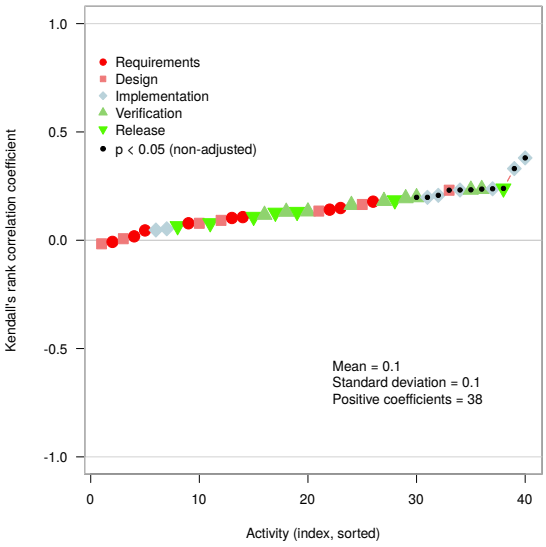


Figure 9: Correlations between the degrees of agility and use of security engineering activities

The magnitudes of the coefficients are small, however. Because all of the security engineering variables measure essentially the same phenomenon, a Bonferroni correction is also justified. After this correction is applied, only two of the coefficients are statistically significant ($p < 0.05 \,/\, 40$). These are coding standards and security reviews with automated tools. The first of these is spurious: coding standards are included as a specific item in the question about agile practices, and, therefore, it is also a part of the combined agility variable. The second one holds, indicating a higher degree of agility increases the use of automated tools for security reviews. This observation makes it reasonable to *tentatively* conclude that particularly agile software development organizations may succeed better in the adaptation of security engineering practices into their existing agile development processes. High degree of professionalism, inherent in organizations utilizing security engineering, may also be a factor in adaptation of advanced agile practices.

Table 5: Statistically significant correlations between the degrees of agility and use of security engineering activities

Activity	Phase	τ-coefficient	p-value
Penetration testing	Verification	0.198	0.045
Static analysis	Implementation	0.198	0.046
Security specific hardening sprints	Implementation	0.207	0.042
Architecture and application development guidelines	Design	0.231	0.019
External interface review	Implementation	0.232	0.012
Development time process security audits	Verification	0.233	0.021
Code reviews during testing	Verification	0.236	0.017
Deprecation of unsafe functions	Implementation	0.238	0.015
Ensure host and network security basics are in place	Release	0.239	0.019
Coding standards	Implementation	0.331	0.001
Security reviews with automated tools	Implementation	0.380	< 0.001

5. Discussion

Three research questions were posed in Chapter 3 to gauge the objective of understanding the relationship between software engineering and security engineering practices, and their reciprocal impact. A questionnaire survey was designed to collect empirical data which was presented and analyzed in Chapter 4. This section discusses the attained results in more detail, by going over the key results directly associated to the research questions, implications for the industry, and, implications for the research community. Afterwards, possible limitations and caveats affecting the survey and attained results are noted prior to closing conclusions.

5.1. Key results

To summarize, the benefits of agile methodologies have been realized in all aspects of software development, and security critical areas are not an exception to this general rule. The degree of agility tends to grow along the systematic use of security engineering activities. Also, the higher the degree of agility is reported, the more specialized the security engineering activities get. The results also show an increase in the use of agile practices in organizations facing formal security requirements. Another essential finding was the perceived impact of the security engineering practices which in most cases correlated negatively with their use. Some of the most efficient software security engineering practices receive much less usage than their impact suggests.

For each research question, we note the key results to be the following:

RQ1: Security engineering in software development is dominated by agile methodologies and activities. Agile software development focuses on requirement engineering, implementation and testing, whereas the focus of security engineering appears to be in verification when judged by the number of suggested techniques. Based on the results, however, implementation-time security engineering practices were the most used, appearing to correlate well with the extant software engineering practices. Use of verification techniques heavily concentrated into few key practices.

RQ2: Perceived impact versus use of security engineering activities was directly asked from the respondents. This was analyzed by comparing the frequency of use of a security engineering practice to its perceived impact, producing some interesting outliers. The most basic security engineering practices were deemed effective, and were also most used; however, the more specialized security practices received much less usage than their perceived effect would suggest.

RQ3: Correlations between agile practices and security engineering activities is positive: degree of agility grows with increased use of security engineering practices. Formal security requirements tend to increase the level of agility. This analysis also supports the findings of RQ1, with significantly increased use of implementation and verification phase security engineering practices in the most agile organizations.

5.2. Implications for the industry

The survey was conducted to provide information about the established practices of software security engineering. 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 [49].

The findings suggest that security engineering in software development is mostly performed by experienced and educated personnel – implying adeptness – in various organizations, especially producing web and cloud software exposed to the Internet. Agile methods are prevalent also in security work, and not only that: they are most utilized in the most security-intensive projects.

Regulation is a main driving force in security work, although not the only one, and the results imply a need for improvement in the mechanisms how security regulation is enforced. Especially the security assurance regarding

security policies and security enforcement mechanisms appears a bottleneck. 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. The aim of regulation should be improvement of software security, not conforming with formalities. Development of security regulation itself might benefit from a more frequent update and release cycle.

On the other hand, the scheduling and resourcing pressures of the software development projects also affect the application of security practices. Tools, and tool-based activities (automated security tests, fuzz testing, dynamic analyses) are prevalently underused. Efficient but very rarely used hardening sprints are an extreme example of this. An increasing amount of organizations dedicate work and even whole sprints to technical debt [23], but only a small minority of 7% does the same for security.

5.3. Implications for the research community

The activities contained in the Software Security Life Cycle models are almost universally utilized; however, the models themselves are rarely, if at all, adapted as a whole. They remain a mere security engineering framework, from which the practitioners pick and choose the practices deemed most suitable for their work. However, as the impact measure reveals, use and perceived impact of the practices correlate quite poorly. This can be attributed to several possible factors, but the main conclusion based on this observation is two-fold: firstly, software development organizations should be more proactive in updating their processes, but, secondly, the research community working on software security development models and certificates should pay high attention to their adaptability.

Further, a trend in the application of specific security activities to software development can be observed from the results presented in Section 4.2: security activities executable as part of normal software development activities are more systematically used. For example, "Establish security requirements" can be done as part of Scrum's backlog formation, or the use of "Coding standards" and "Code reviews during testing" are in line with agile principles and notes on software craftsmanship [36].

This is an expected observation, but one which implicates targets that require further examination. Firstly, it should be established which security activities are synergic (i.e. the security activity can be implemented more efficiently as part of a software-development-practice-defined activity). Findings from here could be used to motivate and better design security-built-in software development practices in the future.

Second, are the synergic security activities used solely to provide security for the software projects, or do they require additional–explicit–security activities to retain adequate levels of security in the project. The results can be used to argue for or against the usage efficiency of synergic security activities: secure software development life-cycles have linked activities for the reason of ensuring exhaustive hardening; organization may use singular activities without this knowledge and attain significantly lower levels of security than expected.

Third, contrasting the synergic and non-synergic security activities, it should be possible to understand the impact on software development efficiency. For example, for the sake of arguing for or against the use of particular set of self-defined security constraints, the development organization should have knowledge about the level of security attained and the development efficiency attainable via the use of particular set of security activities.

To facilitate the three points described above, further empirical data needs to be gathered. This includes both quantitative development efficiency data, and static security breach analysis data, as well as qualitative data on the stakeholder's argumentation, relating to how security is defined and attained in their software projects. Direct interviews with the security practitioners, combined with a series of industry surveys for various target groups will provide an excellent overview of the specific best practices and the needs for further development. The studies should also be conducted in a wider than a national scope within one country to allow comparison.

5.4. Limitations and caveats

Surveys have inherent non-response and sampling frame biases, and produce self-reported results. With these limitations, this survey illuminates the state of art in an area not previously researched in the selected context. In this particular survey, several precautions were taken to mitigate the limitations. First, the respondent's relevant background details were asked extensively to ensure the external validity. On this subject, two things were noted: (a) the respondent profile appears very similar to other recent surveys conducted in the same industry (software development) and country (Finland); (b) the respondents answering this survey, *de facto* software security practitioners, were somewhat biased towards the most experienced and educated groups. This may also be taken to represent the actual situation in the industry.

To obtain quantitative and analyzable data, the answer options were necessarily limited to a set of predefined values. Certain answer options were derived from previous industry surveys to achieve better, or in some cases even direct comparability. In the security engineering part of the questionnaire, the answer options were selected from security engineering life cycle models and standards claiming industry prevalence. Many of these claims not directly validated in this study, and in general industry context. The amount of agile and security practices was deemed quite near the upper limit of practically answerable, 56

options in 7 questions, accompanied by 5 optional (usage-based) questions for the 40 options for security impact (see survey design in Chapter 3.3). These questions were noted to be the part where most non-respondents stopped answering, based on the statistics provided by the survey service. Even this extensive set of practices does not, however, guarantee that the entire field of security engineering practices were covered: some industry best practises, created by practical experience, may have left unrevealed. Furthermore, the validity of the survey's construct was subject to a necessary compromise between usability and thoroughness. Open text fields were used extensively, and the lack of free-text additions to the answering options is taken to support the selections made in the survey's design. Overall feedback from the recipients ranged from neutral to positive. Finally, the revealed correlation between agility and security practices does not imply causality, nor does it exclude external factors possibly affecting both. This makes several hypotheses possible, to be tested in future research.

5.5. Conclusions

This article reports the findings of an industry survey about software security engineering practises used in conjunction with agile software development practises. The survey was conducted among software development practitioners in Finland in early 2018.

The use of these practises was surveyed using precompiled lists using a five level Likert scale. The queried security engineering practices were identified from several software security life cycle models; agile practises were derived from previous industry studies. For each used security engineering practise, the respondents perception of its security impact was asked. This produces a clear and unambiguous picture of the current use of security engineering in agile software development, and the perceived effect on the software security, presented in Chapter 4.

Agile practices are prevalent among the respondents. Introduction of security practices appears to correlate with increased use of agile. Extant software security development life cycle models do not appear to guide the security engineering work: instead, practices appear to be selected in conjunction with the software development practices.

Security engineering in software development conforms with agile practices. Agile usage correlates with usage of security engineering: when the use of activities categorized as security engineering practices increases, the use of agile practices is also increased. The use of security engineering is concentrated in implementation phase. Results show a discrepancy between the usage and perceived security impact of the activities, indicating a requirement for better integration of security engineering methods and tools into the agile software development processes and toolchains.

References

[1] 24748-1:2018, I., 2018. Systems and software engineering – Life cycle management – Part 1: Guidelines for life cycle management, 1st Edition. ISO/IEC/IEEE.

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

[3] Avizienis, A., Laprie, J. C., 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.

[4] 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

[5] 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

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

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

[8] 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.

[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] Hamid, B., Weber, D., 2018. Engineering secure systems: Models, patterns and empirical validation. Computers & Security 77, 315–348.
[22] 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.
[23] 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.
[24] Howard, M., Lipner, S., 2006. The security development lifecycle. Vol. 8. Microsoft Press Redmond.
[25] 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
[26] IEEE, 2018. Avoiding the top 10 software security design flaws.
[27] ISO/IEC standard 15026-4:2012, 2012. Systems and software engineering – Systems and software assurance – Part 4: Assurance in the life cycle. ISO/IEC.
[28] ISO/IEC standard 15408-1:2009, 2014. Information technology - Security techniques - Evaluation criteria for IT security, 3rd Edition. ISO/IEC.
[29] ISO/IEC standard 21827:2008, 2008. Information Technology – Security Techniques – Systems Security Engineering – Capability Maturity Model (SSE-CMM), 2nd Edition. ISO/IEC.
[30] KATAKRI 2015, 2015. Information security audit tool for authorities. Referenced 12 Nov. 2018.
URL https://www.defmin.fi/files/3417/Katakri_2015_Information_security_audit_tool_for_authorities_Finland.pdf
[31] Kitchenham, B. A., Pfleeger, S. L., Jan. 2002. Principles of survey research part 2: Designing a survey. SIGSOFT Softw. Eng. Notes 27 (1), 18–20.
URL http://doi.acm.org/10.1145/566493.566495
[32] Klees, G., Ruef, A., Cooper, B., Wei, S., Hicks, M., Aug. 2018. Evaluating Fuzz Testing. ArXiv e-prints.
[33] 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
[34] 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
[35] 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.
[36] Martin, R. C., 2009. Clean code: a handbook of agile software craftsmanship. Pearson Education.
[37] McDermott, J., Dec 2001. Abuse-case-based assurance arguments. In: Seventeenth Annual Computer Security Applications Conference. pp. 366–374.
[38] 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.
[39] McGraw, G., 2006. Software Security: Building Security In. Addison-Wesley Professional.
[40] Microsoft, 2017. Agile development using microsoft security development lifecycle.
[41] 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.
[42] 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
[43] 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
[44] NIST, 2018. Source code security analyzers.
[45] OWASP, 2018. Owasp top 10 application security risks.
[46] OWASP SAMM, 2017. Software assurance maturity model.
[47] 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.
[48] Pfleeger, S. L., Kitchenham, B. A., Nov. 2001. Principles of survey research: Part 1: Turning lemons into lemonade. SIGSOFT Softw. Eng. Notes 26 (6), 16–18.
URL http://doi.acm.org/10.1145/505532.505535
[49] 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.
[50] 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).
[51] 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
[52] 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.
[53] 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
[54] Rodríguez, P., Markkula, J., Oivo, M., Turula, K., Sept 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. pp. 139–148.
[55] SANS, 2011. CWE/SANS top 25 most dangerous software errors.
[56] Savola, R. M., 2017. Current level of cybersecurity competence and future development: Case finland. In: Proceedings of the

11th European Conference on Software Architecture: Companion Proceedings. ECSA '17. ACM, New York, NY, USA, pp. 121–124.
URL http://doi.acm.org/10.1145/3129790.3129804

[57] Scandariato, R., Wuyts, K., Joosen, W., 2015. A descriptive study of microsoft's threat modeling technique. Requirements Engineering 20, 163–180.

[58] Schwaber, K., Beedle, M., 2002. Agile Software Development with Scrum, 1st Edition. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[59] 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.

[60] 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.

[61] 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

[62] Stavru, S., 2014. A critical examination of recent industrial surveys on agile method usage. Journal of Systems and Software 94, 87 – 97.

[63] Synopsys Software Integrity Group, 2017. The building security in maturity model.
URL https://www.bsimm.com/

[64] 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.

[65] Tondel, 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.

[66] Tsipenyuk, K., Chess, B., McGraw, G., 2005. Seven pernicious kingdoms: A taxonomy of software security errors. IEEE Security & Privacy 3 (6), 81–84.

[67] 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.

[68] 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.

[69] VAHTI, 2015. VAHTI-ohje (trans. VAHTI instruction).
URL http://www.vahtiohje.fi/web/guest

[70] VAHTI 1/2013, 2013. Sovelluskehityksen tietoturvaohje (trans. security instructions for software engineering). Referenced 8th Oct. 2017.
URL https://www.vahtiohje.fi/web/guest/vahti-1/2013-sovelluskehityksen-tietoturvaohje

[71] VersionOne, 2018. 12th annual state of agile survey.

[72] Viega, J., McGraw, G., 2002. Building Secure Software: How to Avoid Security Problems the Right Way, 1st Edition. Addison-Wesley.

[73] 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

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

[75] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A., 2012. Experimentation in Software Engineering. Springer Publishing Company, Incorporated.

[76] Yost, J. R., 2007. 20 - a history of computer security standards. In: Leeuw, K. D., Bergstra, J. (Eds.), The History of Information Security. Elsevier Science B.V., Amsterdam, pp. 595 – 621.
URL http://www.sciencedirect.com/science/article/pii/B9780444516084500213

[77] Yost, J. R., Apr 2015. The origin and early history of the computer security software products industry. IEEE Annals of the History of Computing 37 (2), 46–58.

[78] 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.

Turku Centre for Computer Science
TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspnäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, Z_4-Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations Between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen,** On the NTRU Cryptosystem
64. **Saeed Salehi**, Varieties of Tree Languages
65. **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
66. **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
67. **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
68. **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
69. **Luigia Petre**, Modeling with Action Systems
70. **Lu Yan**, Systematic Design of Ubiquitous Systems
71. **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
72. **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
73. **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
74. **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
75. **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
76. **Jussi Salmi**, Improving Data Analysis in Proteomics
77. **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
78. **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
79. **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
80. **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
81. **Luka Milovanov**, Agile Software Development in an Academic Environment
82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Mehtods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming

128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performace Estimation
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
149. **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
150. **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
151. **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
152. **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
153. **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
154. **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip
155. **Mikko Pelto**, On Identifying and Locating-Dominating Codes in the Infinite King Grid
156. **Anton Tarasyuk**, Formal Development and Quantitative Verification of Dependable Systems
157. **Muhammad Mohsin Saleemi**, Towards Combining Interactive Mobile TV and Smart Spaces: Architectures, Tools and Application Development
158. **Tommi J. M. Lehtinen**, Numbers and Languages
159. **Peter Sarlin**, Mapping Financial Stability
160. **Alexander Wei Yin**, On Energy Efficient Computing Platforms
161. **Mikołaj Olszewski**, Scaling Up Stepwise Feature Introduction to Construction of Large Software Systems
162. **Maryam Kamali**, Reusable Formal Architectures for Networked Systems
163. **Zhiyuan Yao**, Visual Customer Segmentation and Behavior Analysis – A SOM-Based Approach
164. **Timo Jolivet**, Combinatorics of Pisot Substitutions
165. **Rajeev Kumar Kanth**, Analysis and Life Cycle Assessment of Printed Antennas for Sustainable Wireless Systems
166. **Khalid Latif**, Design Space Exploration for MPSoC Architectures

167. **Bo Yang**, Towards Optimal Application Mapping for Energy-Efficient Many-Core Platforms
168. **Ali Hanzala Khan**, Consistency of UML Based Designs Using Ontology Reasoners
169. **Sonja Leskinen**, m-Equine: IS Support for the Horse Industry
170. **Fareed Ahmed Jokhio**, Video Transcoding in a Distributed Cloud Computing Environment
171. **Moazzam Fareed Niazi**, A Model-Based Development and Verification Framework for Distributed System-on-Chip Architecture
172. **Mari Huova**, Combinatorics on Words: New Aspects on Avoidability, Defect Effect, Equations and Palindromes
173. **Ville Timonen**, Scalable Algorithms for Height Field Illumination
174. **Henri Korvela**, Virtual Communities – A Virtual Treasure Trove for End-User Developers
175. **Kameswar Rao Vaddina**, Thermal-Aware Networked Many-Core Systems
176. **Janne Lahtiranta**, New and Emerging Challenges of the ICT-Mediated Health and Well-Being Services
177. **Irum Rauf**, Design and Validation of Stateful Composite RESTful Web Services
178. **Jari Björne**, Biomedical Event Extraction with Machine Learning
179. **Katri Haverinen**, Natural Language Processing Resources for Finnish: Corpus Development in the General and Clinical Domains
180. **Ville Salo**, Subshifts with Simple Cellular Automata
181. **Johan Ersfolk**, Scheduling Dynamic Dataflow Graphs
182. **Hongyan Liu**, On Advancing Business Intelligence in the Electricity Retail Market
183. **Adnan Ashraf**, Cost-Efficient Virtual Machine Management: Provisioning, Admission Control, and Consolidation
184. **Muhammad Nazrul Islam**, Design and Evaluation of Web Interface Signs to Improve Web Usability: A Semiotic Framework
185. **Johannes Tuikkala**, Algorithmic Techniques in Gene Expression Processing: From Imputation to Visualization
186. **Natalia Díaz Rodríguez**, Semantic and Fuzzy Modelling for Human Behaviour Recognition in Smart Spaces. A Case Study on Ambient Assisted Living
187. **Mikko Pänkäälä**, Potential and Challenges of Analog Reconfigurable Computation in Modern and Future CMOS
188. **Sami Hyrynsalmi**, Letters from the War of Ecosystems – An Analysis of Independent Software Vendors in Mobile Application Marketplaces
189. **Seppo Pulkkinen**, Efficient Optimization Algorithms for Nonlinear Data Analysis
190. **Sami Pyöttiälä**, Optimization and Measuring Techniques for Collect-and-Place Machines in Printed Circuit Board Industry
191. **Syed Mohammad Asad Hassan Jafri**, Virtual Runtime Application Partitions for Resource Management in Massively Parallel Architectures
192. **Toni Ernvall**, On Distributed Storage Codes
193. **Yuliya Prokhorova**, Rigorous Development of Safety-Critical Systems
194. **Olli Lahdenoja**, Local Binary Patterns in Focal-Plane Processing – Analysis and Applications
195. **Annika H. Holmbom**, Visual Analytics for Behavioral and Niche Market Segmentation
196. **Sergey Ostroumov**, Agent-Based Management System for Many-Core Platforms: Rigorous Design and Efficient Implementation
197. **Espen Suenson**, How Computer Programmers Work – Understanding Software Development in Practise
198. **Tuomas Poikela**, Readout Architectures for Hybrid Pixel Detector Readout Chips
199. **Bogdan Iancu**, Quantitative Refinement of Reaction-Based Biomodels
200. **Ilkka Törmä**, Structural and Computational Existence Results for Multidimensional Subshifts
201. **Sebastian Okser**, Scalable Feature Selection Applications for Genome-Wide Association Studies of Complex Diseases
202. **Fredrik Abbors**, Model-Based Testing of Software Systems: Functionality and Performance
203. **Inna Pereverzeva**, Formal Development of Resilient Distributed Systems
204. **Mikhail Barash**, Defining Contexts in Context-Free Grammars
205. **Sepinoud Azimi**, Computational Models for and from Biology: Simple Gene Assembly and Reaction Systems
206. **Petter Sandvik**, Formal Modelling for Digital Media Distribution

207. **Jongyun Moon**, Hydrogen Sensor Application of Anodic Titanium Oxide Nanostructures
208. **Simon Holmbacka**, Energy Aware Software for Many-Core Systems
209. **Charalampos Zinoviadis**, Hierarchy and Expansiveness in Two-Dimensional Subshifts of Finite Type
210. **Mika Murtojärvi**, Efficient Algorithms for Coastal Geographic Problems
211. **Sami Mäkelä**, Cohesion Metrics for Improving Software Quality
212. **Eyal Eshet**, Examining Human-Centered Design Practice in the Mobile Apps Era
213. **Jetro Vesti**, Rich Words and Balanced Words
214. **Jarkko Peltomäki**, Privileged Words and Sturmian Words
215. **Fahimeh Farahnakian**, Energy and Performance Management of Virtual Machines: Provisioning, Placement and Consolidation
216. **Diana-Elena Gratie**, Refinement of Biomodels Using Petri Nets
217. **Harri Merisaari**, Algorithmic Analysis Techniques for Molecular Imaging
218. **Stefan Grönroos**, Efficient and Low-Cost Software Defined Radio on Commodity Hardware
219. **Noora Nieminen**, Garbling Schemes and Applications
220. **Ville Taajamaa**, O-CDIO: Engineering Education Framework with Embedded Design Thinking Methods
221. **Johannes Holvitie**, Technical Debt in Software Development – Examining Premises and Overcoming Implementation for Efficient Management
222. **Tewodros Deneke**, Proactive Management of Video Transcoding Services
223. **Kashif Javed**, Model-Driven Development and Verification of Fault Tolerant Systems
224. **Pekka Naula**, Sparse Predictive Modeling – A Cost-Effective Perspective
225. **Antti Hakkala**, On Security and Privacy for Networked Information Society – Observations and Solutions for Security Engineering and Trust Building in Advanced Societal Processes
226. **Anne-Maarit Majanoja**, Selective Outsourcing in Global IT Services – Operational Level Challenges and Opportunities
227. **Samuel Rönnqvist**, Knowledge-Lean Text Mining
228. **Mohammad-Hashem Hahgbayan**, Energy-Efficient and Reliable Computing in Dark Silicon Era
229. **Charmi Panchal**, Qualitative Methods for Modeling Biochemical Systems and Datasets: The Logicome and the Reaction Systems Approaches
230. **Erkki Kaila**, Utilizing Educational Technology in Computer Science and Programming Courses: Theory and Practice
231. **Fredrik Robertsén**, The Lattice Boltzmann Method, a Petaflop and Beyond
232. **Jonne Pohjankukka**, Machine Learning Approaches for Natural Resource Data
233. **Paavo Nevalainen**, Geometric Data Understanding: Deriving Case-Specific Features
234. **Michal Szabados**, An Algebraic Approach to Nivat's Conjecture
235. **Tuan Nguyen Gia**, Design for Energy-Efficient and Reliable Fog-Assisted Healthcare IoT Systems
236. **Anil Kanduri**, Adaptive Knobs for Resource Efficient Computing
237. **Veronika Suni**, Computational Methods and Tools for Protein Phosphorylation Analysis
238. **Behailu Negash**, Interoperating Networked Embedded Systems to Compose the Web of Things
239. **Kalle Rindell**, Development of Secure Software: Rationale, Standards and Practices

Turku
Centre *for*
Computer
Science

http://www. tucs.fi

tucs@abo.fi

University of Turku
Faculty of Science and Engineering
- Department of Future Technologies
- Department of Mathematics and Statistics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Faculty of Science and Engineering
- Computer Engineering
- Computer Science

Faculty of Social Sciences, Business and Economics
- Information Systems

ISBN 978-952-12-3833-8
ISSN 1239-1883

Kalle Rindell

Development of Secure Software

Kalle Rindell

Development of Secure Software

Kalle Rindell

Development of Secure Software: Rationale, Standards and Practices