Table of Contents

International Journal of Secure Software Engineering

Volume 8 • Issue 1 • January-March-2017 • ISSN: 1947-3036 • eISSN: 1947-3044

An official publication of the Information Resources Management Association

Editorial Preface

Research Articles

COPYRIGHT

The *International Journal of Secure Software Engineering* is indexed or listed in the following: ACM Digital Library; Bacon's Media Directory; Cabell's Directories; DBLP; Google Scholar; INSPEC; JournalTOCs; MediaFinder; The Standard Periodical Directory; Ulrich's Periodicals Directory

International Journal of Secure Software Engineering
Volume 8 • Issue 1 • January-March 2017

A Lightweight Measurement of Software Security Skills, Usage and Training Needs in Agile Teams

Tosin Daniel Oyetoyan, Department of Software Engineering, Safety & Security, SINTEF Digital, Trondheim, Norway

Martin Gilje Jaatun, Department of Software Engineering, Safety & Security, SINTEF Digital, Trondheim, Norway

Daniela Soares Cruzes, Department of Software Engineering, Safety & Security, SINTEF Digital, Trondheim, Norway

ABSTRACT

Although most organizations understand the need for application security at an abstract level, achieving adequate software security at the sharp end requires taking bold steps to address security practices within the organization. In the Agile software development world, a security engineering process is unacceptable if it is perceived to run counter to the agile values, and agile teams have thus approached software security activities in their own way. To improve security within agile settings requires that management understands the current practices of software security activities within their agile teams. In this study, the authors have used a survey instrument to investigate software security usage, competence, and training needs in two agile organizations. They find that (1) The two organizations perform differently in terms of core software security activities, but are similar when secondary activities that could be leveraged for security are considered (2) regardless of cost or benefit, skill drives the kind of activities that are performed (3) Secure design is expressed as the most important training need by all groups in both organizations (4) Effective software security adoption in agile setting is not automatic, it requires a driver.

KEYWORDS

Agile Software Development, Empirical Study, Software Security, Software Security Activities

1. INTRODUCTION

Protecting the organization's assets from security threats is vital. Security cannot be treated as an add-on functionality or isolated product feature (Gary McGraw, 2006), and it is thus important that security is "built-in" in the process and the product. However, a traditional security engineering process is often associated with additional development efforts and is likely to invoke resentment among agile development teams (ben Othmane et al., 2014; Beznosov & Kruchten, 2004). A software security approach tailored to the agile mind-set thus seems necessary.

Some approaches have been proposed to integrate security activities into agile development, e.g., the Microsoft SDL for Agile (Microsoft, 2012). However, these approaches have been criticised for looking similar to the traditional versions in terms of workload (e.g., performing a long list of security verification and validation tasks) (ben Othmane et al., 2014). As a result, "agile" organizations have approached software security in a way that fits their process and practices. Statistics show that more than 70% of reported vulnerabilities are in the application layer (Fong & Okun, 2007) and not the network. Thus, regardless of whether agile is perceived to be incompatible with any particular secure software development lifecycle, the major discussion we should have is how to improve security

DOI: 10.4018/IJSSE.2017010101

within the agile context (Bartsch, 2011). Previous studies (Ayalew et al., 2013; Baca & Carlsson, 2011) have investigated which security activities are practiced in different organizations, and which are compatible with agile practices from cost and benefit perspectives. Using a survey of software security activities among software practitioners, they identify and recommend certain security activities that are compatible with agile practices such as; eliciting security requirements, using a role matrix, risk analysis, employing secure design principles, drawing countermeasure graphs, adhering to coding rules, wielding security tools, penetration testing, and operational planning and readiness.

While these activities could be argued to be beneficial and cost effective to integrate, there are still gaps between what is "adequate" security (Allen, 2005), and what is currently practiced within several organizations. According to Allen (2005), adequate security is defined as "The condition where the protection and sustainability strategies for an organization's critical assets and business processes are commensurate with the organization's tolerance for risk."

The research presented here is motivated based on the perceived knowledge gaps in software security in agile software development organizations in Norway (Jaatun et al., 2015). In order to address these gaps, management must first understand the current status of software security practices and capability within their organization. This study is carried out in 2 organizations (in the following referred to as "Org-1" and "Org-2"), that develop software in telecommunication and transportation, respectively (see section 3.2.1 for more information on the two organizations). This paper extends our previous work (Oyetoyan et al., 2016) investigating existing practice, skills, and training needs within agile teams, by significantly expanding the background, exploring new dimensions of the data with additional research questions, and deeper discussion of the results. We want to know more on the training needs and understand the relationships between skills and usage of security activities among teams and across roles. The findings are important to guide management decisions towards improving security within their organization.

The Building Security In Maturity Model (BSIMM) (Gary McGraw et al., 2016) has also been used to measure security practices in different organizations. Jaatun et al. (2015) used a questionnaire based on the BSIMM activities to measure the security maturity of Norwegian public organizations. They found that there is a need for improvements in metrics, penetration testing and training developers in secure development. BSIMM is useful for measuring the software security maturity of an organization and helping them formulate overall security strategy (Gary McGraw et al., 2016). However, it is not perceived as a lightweight measurement tool to directly measure developers' skill or usage of software security activities within a development team. Therefore, together with the two organizations under study, we have jointly developed a lightweight instrument consisting of software security activities from OWASP CLASP, Microsoft SDL and the Cigital Touchpoints (de Win et al., 2009) for this purpose.

The rest of this paper is organised as follows: We describe the background in Section 2. We describe our research methodology and study design in Section 3. We then present and discuss the results in Section 4, and finally we offer our conclusions in Section 5.

2. BACKGROUND

Software security has existed as a distinct field of research for over a decade, and reached prominence with the publication of the book "Software Security" (McGraw, 2006).

2.1. Secure Software Development Lifecycles

A number of Secure Software Development Lifecycles (SSDLs) have been proposed, in the following we briefly introduce to most important ones as they relate to this paper.

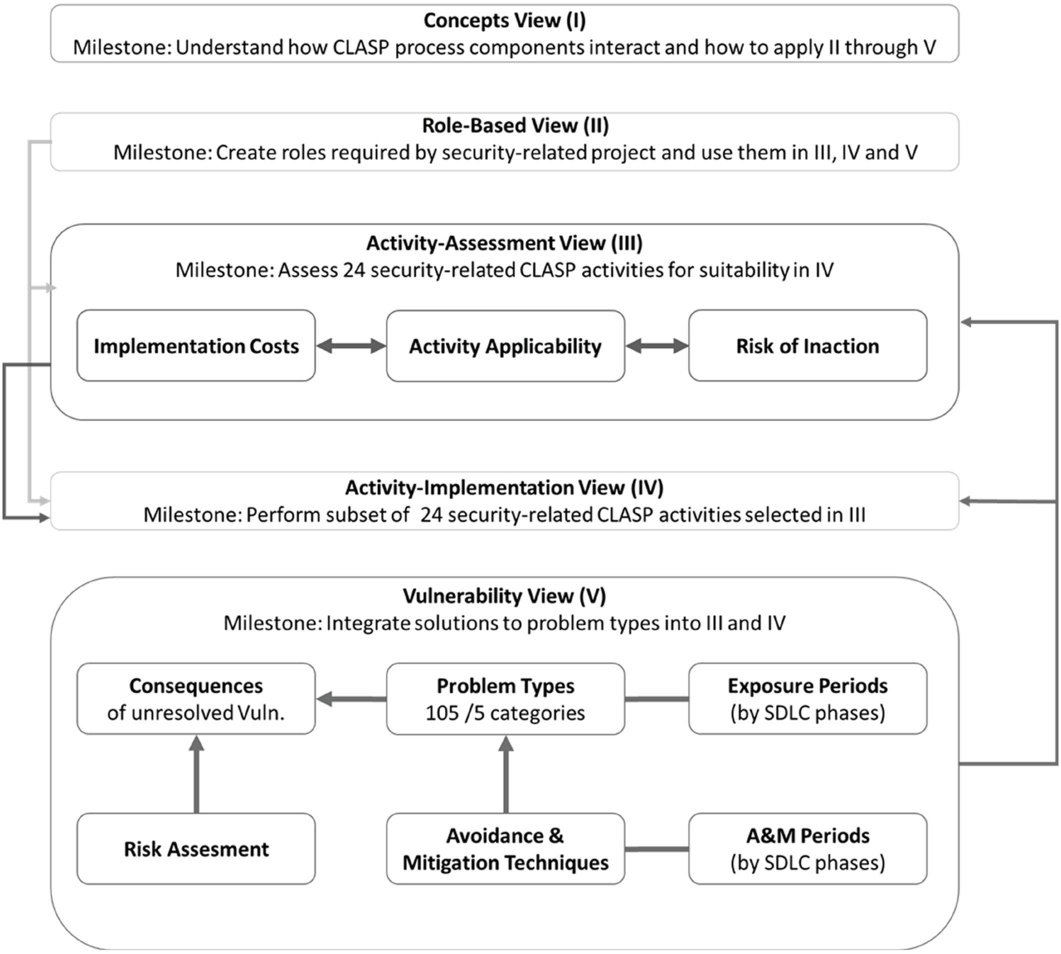
2.1.1. OWASP CLASP

The Comprehensive, Lightweight Application Security Process (CLASP) (OWASP, 2006) was a project under the Open Web Application Security Project (OWASP). A high-level overview of CLASP is given in Figure 1. CLASP was based on seven best practices:

1. Institute awareness programs
2. Perform application assessments
3. Capture security requirements
4. Implement secure development practices
5. Build vulnerability remediation procedures
6. Define and monitor metrics
7. Publish operational security guidelines

CLASP has not been updated since 2006, and is currently considered abandoned. However, some of the CLASP activities can still be considered useful by themselves.

Figure 1. CLASP Overview



Concepts View (I)
Milestone: Understand how CLASP process components interact and how to apply II through V
Role-Based View (II)
Milestone: Create roles required by security-related project and use them in III, IV and V
Activity-Assessment View (III)
Milestone: Assess 24 security-related CLASP activities for suitability in IV
Implementation Costs
Activity Applicability
Risk of Inaction
Activity-Implementation View (IV)
Milestone: Perform subset of 24 security-related CLASP activities selected in III
Vulnerability View (V)
Milestone: Integrate solutions to problem types into III and IV
Consequences of unresolved Vuln.
Problem Types 105 /5 categories
Exposure Periods (by SDLC phases)
Risk Assesment
Avoidance & Mitigation Techniques
A&M Periods (by SDLC phases)

2.1.2. Microsoft SDL for Agile

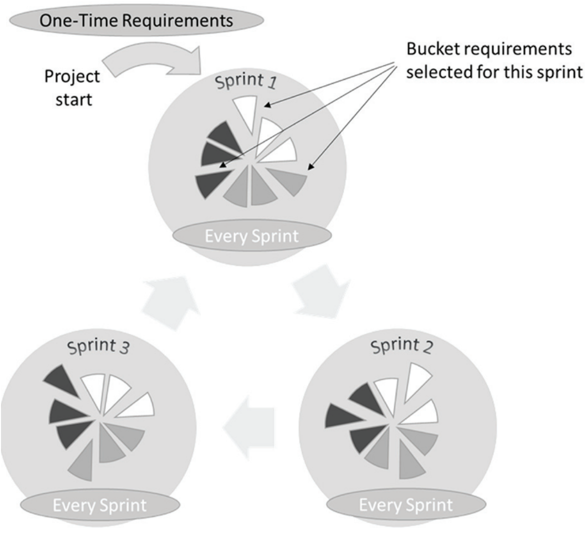
The Microsoft Security Development Lifecycle for Agile Development (SDL-Agile) (Microsoft, 2012) is the agile version of the traditional Microsoft SDL (Howard & Lipner, 2006). SDL-Agile is split into three types of activities (see Table 1);

- "Every-Sprint Requirements" (S) – these activities should be performed in every iteration
- "Bucket Requirements" (B) – these activities must be performed on a regular basis during the development lifecycle; there are three types of such requirements defined (each type referred to as a bucket) and typically one is picked from each bucket in each sprint
- "One-Time Requirements" (O) – these activities are typically only need to be performed once at the beginning of the project (see Figure 2).

Table 1. MS SDL-Agile activities

1. Training	2. Requirement	3. Design	4. Implementation	5. Verification	6. Release	7. Response
1. Core Security Training	2. Establish Security Requirements (O)	5. Establish Design Requirements (O)	8. Use Approved Tools (S)	11. Perform Dynamic Analysis (B)	14. Create an Incident Response Plan (O)	17. Execute Incident Response Plan
	3. Create Quality Bug Bars (B)	6. Perform Attack Surface Analysis/ Reduction (O)	9. Deprecate Unsafe Functions (S)	12. Perform Fuzz Testing (B)	15. Conduct Final Security Review (S)	
	4. Perform Security and Privacy Risk Assessments (O)	7. Use Threat Modeling (S)	10. Perform Static Analysis (S)	13. Conduct Attack Surface Review (B)	16. Certify Release and Archive (S)	

Figure 2. The SDL-Agile One-Time and Bucket Requirements Illustrated



2.1.3. Cigital Touchpoints

The Cigital Touchpoints (Gary McGraw, 2004; Gary McGraw, 2005) were introduced as a lightweight way of distilling the essence of practical software security. They have been presented slightly different over the years, but the essence is as illustrated in Figure 3.

In order of effectiveness, the 7 touchpoints are:

1. Code review
2. Architectural risk analysis
3. Penetration testing
4. Risk-based security tests
5. Abuse cases
6. Security requirements
7. Security operations

2.2. Measuring Software Security Activities

Measuring software security is difficult (Jaatun, 2012), and therefore second-order metrics are often employed, i.e., measuring what kind of software security activities are performed when developing the software.

2.2.1. Open SAMM

The Software Assurance Maturity Model (SAMM or OpenSAMM) (OWASP, 2016) is an open software security framework divided into four business functions: Governance, Construction, Verification and Deployment. Each business function is composed of three security practices, as shown in Table 2.

Each practice is assessed at a maturity level from 1 to 3 (plus 0 for "no maturity"), and for each maturity level there is an objective and two activities that have to be fulfilled to achieve that level. OpenSAMM is "prescriptive", in the sense that it advocates that all the specified activities must be performed in order to be a high-maturity organisation.

2.2.2. BSIMM

The Building Security In Maturity Model (BSIMM) first saw the light of day in 2009, based on a study of 9 software development organizations. BSIMM is structured around a Software Security Framework of four domains, each divided into three practices, as illustrated in Table 3. As is evident from the table, BSIMM shares origins with the OpenSAMM framework described above. The latest

Figure 3. The Cigital Touchpoints

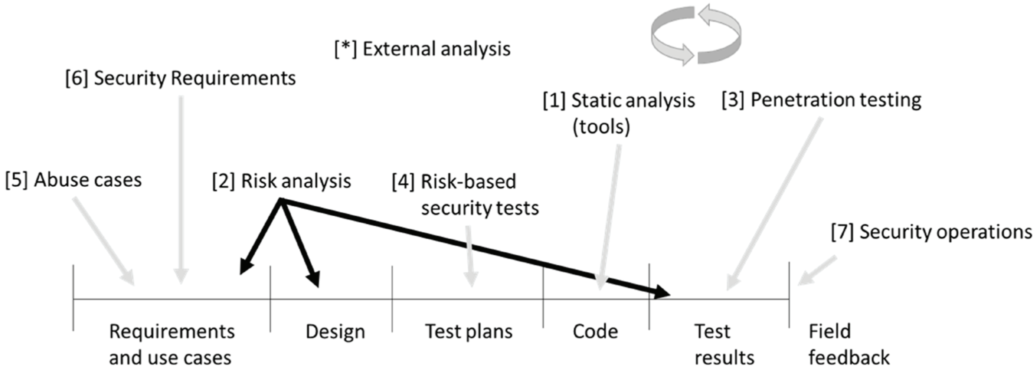


Table 2. The OpenSAMM Software Security Framework

Governance	Construction	Verification	Deployment
Strategy and Metrics	Threat Assessment	Design Review	Vulnerability Management
Policy & Compliance	Security Requirements	Code Review	Environment Hardening
Education & Guidance	Secure Architecture	Security Testing	Operational Enablement

version of the BSIMM report (Gary McGraw et al., 2016) features results from 95 companies, measuring 113 different software security activities.

Although BSIMM also ranks software security activities in three maturity levels, it purports to be descriptive rather than prescriptive, and there is no implicit expectation that all organizations should do all 113 activities. Due to the large number of software security activities, BSIMM can said to be more specific than OpenSAMM. New BSIMM activities are added as they are observed in the field, and activities that fall out of use are removed. The maturity level of a given activity can also be changed from one version of the study to the next.

2.2.3. Common Criteria

The Common Criteria (ISO/IEC) (CC) emerged toward the end of the previous century as an amalgamation of the US DoD Trusted Computer Systems Evaluation Criteria (TCSEC, a.k.a. “the Orange Book”), the European ITSEC and the Canadian CTCPEC. CC is used in the security evaluation of computer-based systems, typically for military or critical infrastructure use. A fundamental concept of CC is that a Protection Profile containing functional security requirements and security assurance requirements is established. There are sets of predefined security assurance requirements which are referred to as Evaluation Assurance Levels (EAL1-7). The manufacturer will create a Security Target document which elaborates how the requirements of the Protection Profile are met, and finally an external evaluator will perform an evaluation to confirm or reject the claims.

CC is essentially a long list of requirements, and it is totally up to the Protection Profile which requirements are considered for a given product. Some of the assurance requirements are effectively software security activities.

2.3. Other Measurement Approaches

We assembled a first list of software security practices as documented in Table 4. This list is taken from previous studies (Ayalew et al., 2013; Baca & Carlsson, 2011). These formed the starting point when developing our survey instrument. The table is an adapted version from Ayalew et al. (2013), and we have merged activities (e.g. attack surface analysis to replace both identify attack surface and attack surface reduction) and also removed some activities (e.g. agree on definitions).

Comparing with the software security activities defined in BSIMM (Gary McGraw et al., 2016), we find that most of the activities in Table 4 are fully or partly covered by BSIMM, except:

Table 3. The BSIMM Software Security Framework

Governance	Intelligence	SSDL Touchpoints	Deployment
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management)

Table 4. Software Security Activities from OWASP CLASP, Microsoft SDL, Cigital Touchpoints (CT) and Common Criteria (CC) and other (O) adapted from (Ayalew et al., 2013; Baca & Carlsson, 2011)

Initial	Requirement	Design	Implementation	Testing	Release
Education (CLASP, SDL)	Security Requirements (CLASP, SDL, CT, CC)*	Risk Analysis (CT, CC)*	Security Tools (SDL)*	Dynamic Analysis (SDL)*	Incident Response Planning (SDL)*
Security Metrics (CLASP)	Quality Gates (SDL)*	Threat Modelling (CLASP, SDL)	Static Code Analysis (SDL, CT)*	Fuzz Testing (SDL)*	Final Security Review (SDL)
	Design Requirements (SDL)	Attack Surface Analysis (SDL)*	Pair programming (O)*	Penetration Testing (CT)*	External Review (CT)
	Abuse cases (CLASP, CT)*	Assumption Documentation (CT)	Coding Rules (SDL)*	Red Team Testing (CT)*	Code Signing (CLASP)
		Critical Asset Analysis (CC)		Risk Based Testing (CT)*	Database Security Configuration (CLASP)
		UMLSec (CC)		Security Code Review (CLASP, SDL)*	Repository Improvement (CC)
		Requirements Inspection (CC)		Security Testing (CLASP)	
		Countermeasure Graph (O)*			
		Cost Analysis (SDL)			
		Security Architecture (CLASP)			
		Secure Design Principles (CLASP)*			
		Identify Trust Boundary (CLASP)			

- UMLSec
- Requirements Inspection
- Countermeasure Graph
- Cost Analysis
- Identify Trust Boundary
- Pair programming

We also note that there are some elements identified by Ayalew et al. (2013) that appear to be duplicates, such as “Penetration Testing” and “Red Team Testing” (the touchpoints do not actually highlight red team testing), and “Risk Based Testing” and “Security Testing” (the touchpoints actually refer to “risk based security tests”). Our final questionnaire is explained in Section 3.2.2, and can be found in full in Appendix A (see Figure 9, Figure 10, and Figure 11).

3. RESEARCH METHODOLOGY AND STUDY DESIGN

The sections below describe the research questions, hypotheses, data collection procedure employed in each case study, the instruments used, and the type of data analysis performed.

3.1. Research Questions

We make the following assumptions that:

- Developers have relatively different skills in software security regardless of the organization where they currently work.
- Agile organizations have different usage patterns with software security activities. An agile team is mostly autonomous and self-confident (Robinson & Sharp, 2004), and thus makes decisions that the team members think best contribute to customer satisfaction and product quality. Since activities are chosen in a voluntary manner in agile settings, we believe that organizations would use activities that best fit their process and business needs.
- Based on conventional wisdom, using an activity requires certain level of know-how. Hence, teams would use activities where they have competence.
- Experienced developers would most probably have taken security related decisions during their development career, and thus have knowledge and experience in software security.

Therefore, we investigate whether the skills, usage and training needs in software security activities in both organizations are similar or different. Understanding the similarities and differences between organizations also help during replications and adoptions of software security activities and programs across different organizations.

The research questions we investigate in this paper are:

RQ1: Which software security activities are most used within the organizations?
RQ2: Which skills are common to both organizations?
RQ3: Which training needs are important to the organizations?
RQ4: How are security experience and the perceived need for software security training influenced by years of developer of experience?
RQ5: What is the relationship between usage of, and skill in software security activities?

3.2. Data Collection

The research performed in this paper is performed in the context of the SoS-Agile project (http://www.sintef.no/sos-agile), which investigates how to meaningfully integrate software security into agile software development activities. The project started in October 2015. The method of choice for the project is Action Research (Greenwood & Levin, 2006). Action research is an appropriate research methodology for this investigation for several reasons. First, the study's combination of scientific and practical objectives is a good match with the basic tenet of action research, which is to merge theory and practice in a way that real-world problems are solved by theoretically informed actions in collaboration between researchers and practitioners (Greenwood & Levin, 2006). Therefore, the design of the instruments had to take in consideration the usefulness of the results for the companies and for research.

Two companies participated closely in the development of the survey questionnaire and have been part of the project since the beginning. In addition, for the interpretation and discussion of the results, the answers from the survey were complemented by document analysis of project artifacts, observations of meetings, and discussions with different stakeholders in the companies. Other focused interviews on specific topics, and the feedback from the survey results, were compared with the collected information about the organizational contexts and documents.

3.2.1. Company Context

This study is carried out in 2 organizations (Org-1 and Org-2) that develop software in telecommunication and transportation, respectively. Both organizations use agile practices (mostly Scrum and a mix of other agile practices) for their software development (see Table 5).

The team set-up for Org-1 (the telecom company) is co-located and distributed; some of the teams are situated in the same location, while some are distributed in different locations and continents. The organization has about 90 engineers. In addition, the telecom company has an information security professional group, as they need to comply with various standards (e.g., PCI DSS (Payment Card Industry, 2016)). The security group also coordinates security audit reports and follow-up on reported security issues. It is also important to mention that the department we are doing research with has a focus on developing new innovation projects.

Org-2 (the transportation software company) is a small/medium software organization with about 50 developers distributed in three different countries. One security officer is officially designated to work on the security procedures and make the security work systematic.

3.2.2. Survey Questionnaire

The questionnaire was designed in phases, getting feedback from the companies and experts for getting to the final version. The first version of the questionnaire contained questions on different software security activities (asterisked activities in Table 4) from OWASP CLASP, Microsoft SDL for Agile, Common Criteria, and Cigital Touchpoints that have been used in previous studies (Ayalew et al., 2013; Baca & Carlsson, 2011) (see Table 4). The table also includes additional practices such as "pair programming" and "drawing a countermeasure graph" considered in these studies; both are common security activities used in agile settings, e.g., when security experts rotate through programming pairs (Bartsch, 2011; Wäyrynen et al., 2004).

The instrument has been jointly reviewed by the authors, a security professional, a security champion and a project manager. The activities are classified differently than in the traditional software development lifecycle (SDLC), but they do, however, fit into each development lifecycle. The rationale is to invoke a different way of perceiving these activities than from a traditional viewpoint. This could make it possible to spot some assumptions such as for instance, whereas secure design involves many activities from "Threat modelling and risk management", we can argue that software designers could make assumptions about secure design when they include, e.g., authentication mechanisms (Arce et al., 2014). However, performing a comprehensive threat analysis could reveal an insecure design, e.g., a possibility to bypass an authentication or authorization mechanism by directly navigating to an obscure webpage or resource.

Similarly, we have considered software security tools separately in order to identify strong and weak areas of usage and skills. Findings from the survey can trigger further questions, e.g., why certain implemented tools are not used within the organization, and this could lead to useful actions. These activities are divided into: Inception, threat modelling and risk management, secure design and

Table 5. Demography of organizations

	Org-1	Org-2
Dominant Agile Process	Scrum (77%)	Scrum (83%)
Industry	Telecom	Transport
Team setup	Co-located/Distributed	Distributed
Have Security Professionals	Yes	No
No. of Respondents	56	36

coding, security tools, security testing, and release. Table 6 shows the software security activities. In addition, we provided a short explanation of each term we have used in the survey for the respondents. We have used a scale for the skill level as shown in Table 7; the respondents were instructed to use this scale when assessing their own skill level.

For the software activities listed in Table 6, we asked the following 3 questions:

Q1 What is your skill level in this activity or tool?
Q2 Do you currently use this activity or tool? (Check box for yes)
Q3 Do you want to have training in this activity or tool? (Check box for yes)

In addition, we asked 2 questions about security and development experience:

Q4 Do you have security experience? (Yes or no)
Q5 Number of years with software development.

We designed both an online questionnaire and a paper-based version. We further refined the instrument by running a test on our industrial contacts, an independent architect and a post-doctoral fellow in software engineering. The target response time was 10-12 minutes. For the most part, the questionnaire was manually administered to the development teams on site. This increased the response rate, and provided the opportunity to clarify questions that respondents might have.

The final questionnaire and the explanation sheets are in the Appendix A.

3.3. Method of Analysis

Our unit of analysis is based on role within the agile team. These roles are developers, architects and testers. To analyse the relationship between skill (Q1) and usage (Q2), we use correlation analysis to find the relationship between both variables. In the case of Q1, which is an ordinal variable, we use the mean value (Knapp, 1990) as an interval variable to statistically compare the skill levels between the organizations. In addition, we take the proportion (ratio) of skill level between "moderate" and "expert" and compare among roles in the organizations.

To find the correlation between security experience (Q4) - nominal variable (Yes/No) and years of development (Q5) - interval variable, we group the interval variable (Q5) into ranges (e.g., 0-5, 6-10, > 10). For each group, we count and record the number of "yes" and "no" for Q4. We then find the ratio of "yes" and "no" for each category. We then used Pearson correlation to find the relationship between the computed ratios. A significant negative correlation implies that the higher the number of years with software development among respondents in the group, the lesser the number with no experience with security.

To assess how years of experience may influence training needs, we group the responses into two categories: 0-5years and >5 years. We then use Wilcoxon test to assess whether there is significant difference between the two groups. Finally, we use a ratio scale for the usage (Q2) and training needs (Q3) for the various software activities among teams and roles. For similarity analysis between organizations, we use Pearson and Spearman correlation tests. We have used the R statistical package[1] for the analysis.

3.4. Assessment of Reliability and Validity

For this type of study, we are concerned about reliability, content validity, and construct validity (Dybå, 2000). We ignore criterion validity because we are not concerned about predictions of any variable, rather we discuss external validity which is related to generalization of the results in this study.

Reliability concerns the degree to which an instrument will produce the same result if it is administered again. One major factor that could make the instrument unreliable in our case is the

Table 6. Software Security Activities in the survey

Inception
Functioning as a project security officer/champion
Establishing security requirements
Writing abuse stories/cases
Addressing security logistics (e.g., creating relevant security fields)
Threat Modeling & Risk Management
Threat Modeling
Attack surface analysis
Countermeasure techniques
Assets analysis
Risk analysis
Role matrix identification
Release
Incident response management
Secure Design & Coding
Secure design (attack surface reduction, secure defaults)
Secure coding (secure guidelines)
Pair programming
Static code analysis
Use of Security tools
Threat modelling tool
Dynamic code analysis tool
Static code analysis tool
Code review tool
Security testing
Vulnerability assessment
Penetration testing
Red team testing
Fuzz testing
Dynamic testing
Risk-based security testing
Security code review

Table 7. Scale for skill level

Novice [1]	Basic [2]	Moderate [3]	High [4]	Expert [5]
Have no experience working in this area	You have the level of experience gained in a classroom and/or experimental scenarios or as a trainee on-the-job. You are expected to need help when performing in this area	You are able to successfully complete tasks in this area as requested. Help from an expert may be required from time to time, but you can usually perform the skill independently	You can perform the actions associated in this area without assistance. You are certainly recognized within your immediate organization as "a person to ask" when difficult questions arise regarding this area	You are known as an expert in this area. You can provide guidance, troubleshoot and answer questions related to this area of expertise and the field where the skill is used

Table 8. Mapping of Software Security Activities

	CLASP	MS-SDL	CT	CC	Others
Functioning as project security officer/champion	*	*			
Gathering security requirements	*	*	*	*	
Writing abuse stories/cases	*		*		
Threat modeling	*	*			
Attack surface analysis	*	*			
Countermeasure techniques	*	*			
Asset analysis				*	
Risk analysis	*		*	*	
Role matrix identification	*				
Secure design	*	*	*		
Secure coding	*	*	*		
Pair programming					*
Static code analysis	*	*	*		
Threat modeling tool					
Dynamic code analysis tool					
Static code analysis tool					
Code review tool					
Vulnerability assessment					
Penetration testing			*		
Red team testing					
Fuzz testing		*			
Dynamic testing	*				
Risk-based testing			*		
Security code review	*		*		
Incident response management	*	*			

respondent's interpretation of the software security activities. To avoid misunderstanding and provide equal level of reasoning about the terms, we have provided a description of the security activities for the respondents.

Content validity is concerned with how adequately the items in the instrument represent the domain of the concept being studied. The items on the instrument we have used are drawn from established software security activities (de Win et al., 2009) (e.g. OWASP CLASP, Microsoft SDL, and Touchpoints). Thus, content validity is built into the study from the beginning.

Construct validity is about whether the instrument really measures what it claims to measure. Determining the skill level (Q1) of a respondent on an activity could lead to erroneous results. For instance, someone with a basic skill level could over-estimate their own ability and assume an average skill. To mitigate this threat, we have used a scale (see Table 7) with a description for each scale category. Construct validity could also be an issue with Q4, since we have not used any description to help the respondent clarify his/her security experience. However, we think that respondents that say "yes" to this question must have had to take security related decisions during their job.

This study involves 2 small/medium sized organizations that are different in the type of market targets for their solutions. Both practiced agile development and have their development teams co-located and distributed. However, we cannot assume generalization of results across different organizations that use agile. There are other contexts that could make the result different or similar. Examples are the type of development model (e.g. outsourcing development model) or the type of software or market targets. More studies would be useful to see how correlated the different organizations are to the ones we have studied.

4. RESULTS

We present the results of the survey and analysis conducted among the two organizations, discussing each research question in turn.

4.1. RQI: Which Activities are Most Used Within the Organizations?

In Figure 4, Figure 5 and Figure 6 we report activities where current usage is 40% or more. Among developers in Org-1, use of code review tool, static code analysis, pair programming and secure coding (40.5%) are the most used activities. Similarly, in Org-2, use of code review tool, static code analysis (with or without tool), and pair programming are the most used activities. 63.4% of developers in Org-1 and 48% of developers in Org-2 say they are confident (moderate to expert skills) in writing secure code. This is slightly higher when compared to research performed by Microsoft (Adams, 2012) where only 36% of developers are confident to write secure software. However, in terms of current practice in both organisations under study, only 40.5% in Org-1 and 23.8% in Org-2 currently indicate to engage in secure coding. This finding shows a gap of roughly 60% in Org-1 and 76% in Org-2 among developers.

Among architects in Org-1, use of code review tool, static code analysis, secure coding, countermeasure techniques, and penetration testing are the most used activities. In Org-2, use of code review tool, static code analysis/tool, pair programming, and dynamic code analysis tool are the most used activities. Only 33.3% of architects in Org-1 and 20% in Org-2 indicate to practice secure design. However, we found that architects in Org-1 perform some activities in secure design processes such as using countermeasure techniques (44%), role matrix identification (33%), asset analysis (11%), and performing attack surface analysis (11%). When comparing with Org-2, we found no usage of such secure design activities among the architects.

At 40% usage threshold (see Figure 7), testers in Org-1 use a code review tool, static code analysis, secure coding, a dynamic code analysis tool, and security code review, whereas testers in Org-2 use a code review tool, static code analysis (with or without tool), fuzz testing, dynamic testing, and risk-based testing. We found that testers in Org-2 use testing approaches (e.g. risk-based testing) that are more suitable during high level testing such as integration or system testing.

The usage correlation results (Table 8) show that developers and architects are similar when we take use of code review tool, pair programming, and static code analysis/tool into consideration. However, when those are taken out, the correlation is very weak and not significant. Among the core security activities such as threat modeling, security testing, secure design and coding, the two organizations differ in the level of usage. The frequently used activities can be viewed as a platform that can be leveraged for secure software development. However, they do not necessarily have to be used for it.

The result among developers and architects indicates that Org-1 performs more core security activities than Org-2 (Table 9). For instance, Org-1 does secure design and coding twice as much as Org-2. 16% of the architects in Org-1 perform threat modeling, whereas none of the architects performs this activity in Org-2. The situation is the same for the remaining activities, security testing, requirements and release. The exception is in security testing and among testers, where Org-2 performs higher than Org-1. The obvious reason is that testers in Org-2 perform risk-based testing predominantly (50% usage), because the testing team is independent of the development team.

We interacted with the CISO in Org-1. One reason for higher usage in Org-1 could be attributed to the presence of the security expert group in Org-1, although it does not appear that the security group influences the choice of software security activities that are performed by the teams. The agile teams are very independent, innovative, and make their own decisions. Nevertheless, the level of awareness about security is higher in Org-1. According to the security professional we talked to in Org-1, this awareness among teams is due to tech talks, demos and other community building activities that have allowed the teams to learn from each other, and to develop ideas and ways of doing things across the development teams.

The similarities and differences in usages among architects and testers in both organizations could be explained by the fact that most of the architects in both organizations have a development role. While this also holds for testers in Org-1, it is different for Org-2.

The four more common activities are classified under the activities that can be leveraged to deliver security, but it is another question whether they are leveraged to tackle security, as we can argue that these activities can be used without focusing them strictly on security. For instance, a team may use static code analysis for checking code quality, conformance to style standard, detecting code complexities, and code smells. These do not translate directly to security defects. We can of course make another line of argument that every defect has a level of security risk, however, not every defect is security related. Secure coding standards and focused security test suites such as those published in NIST SARD[2] project are specific for security defects. Pair programming and use of a code review tool may be focused on catching only “conventional” defects (and not security defects). We take this background into consideration in our analysis and discussion.

Table 9. Correlation analysis between organizations wrt skill, usage, and training needs grouped by roles (*= statistically significant at 95% confidence interval)

	Developers		Architects		Testers	
	All Activities	FUA	All Activities	FUA	All Activities	FUA
Skill	0.88*	0.68*	0.82*	0.76*	0.11	-0.084
Usage	0.86*	0.37	0.73*	0.26	0.52*	0.12
Training Needs	0.504*	0.678*	0.497*	0.453	0.064	0.186

4.2. RQ2: Which Skills are Common to Both Organizations?

In this Section, we report the activities where the majority (>= 50%) of respondents indicate to have moderate to expert skills. Among the developers in Org-1 (Figure 4), the majority of respondents have moderate or higher skills in use of code review tool, pair programming, secure coding, secure design, and static code analysis. In Org-2 the situation is relatively similar. The activities/tools are in the order: use of code review tool, pair programming, and static code analysis. Among respondents who function as architects, in Org-1, we found that in addition to the above activities indicated by developers, a majority of the architects indicate strong skill in security code review (73%). In Org-2, a majority of architects in addition indicate to have skills in secure coding, secure design, use of dynamic code analysis tool, and countermeasure techniques. A majority of the testers in Org-1 indicate to have skills in code review, pair programming, secure design, security code review, secure coding, static code analysis, and writing abuse stories/cases; whereas the majority of testers in Org-2 indicate skills in static code analysis, fuzz testing, use of code review tool, dynamic testing, risk analysis, and risk-based testing.

We found strong statistically significant correlations or similarities in the skill level indicated by the developers (0.88) and architects (0.82) in both organizations. The correlation is lower when we removed the four common activities used by both groups. However, the competence areas indicated by testers differ in both organizations, as evidenced by the weak correlation of 0.11. Testers in Org-2 indicate strong competencies in fuzz testing, dynamic testing, risk analysis, and risk-based testing. These are different from testers in Org-1 who indicate stronger competencies in security code review and writing abuse stories.

To understand the result for the tester group, we further investigated the team formation in each of the organization. The distribution shows that 90% of testers in Org-1 are developers while only 40% of testers in Org-2 have a development role. The majority of testers (60%) in Org-2 are therefore independent testers who are not involved in coding. They therefore are more disposed to

Figure 4. Comparison of usage level among developers between the 2 organisations

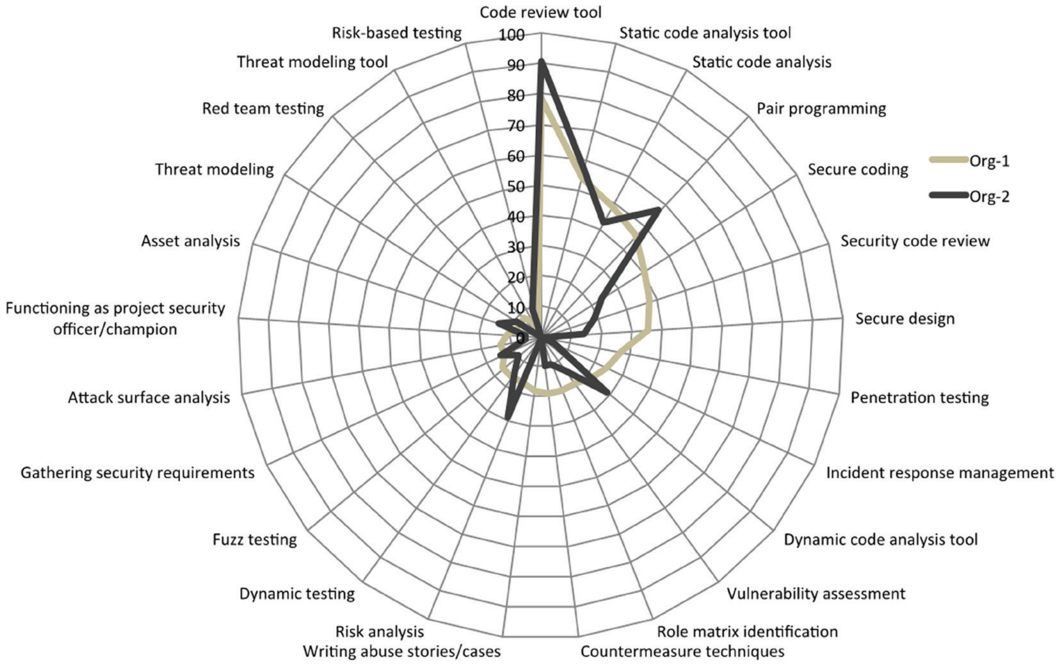


Figure 5. Comparison of usage level among architects between the 2 organisations

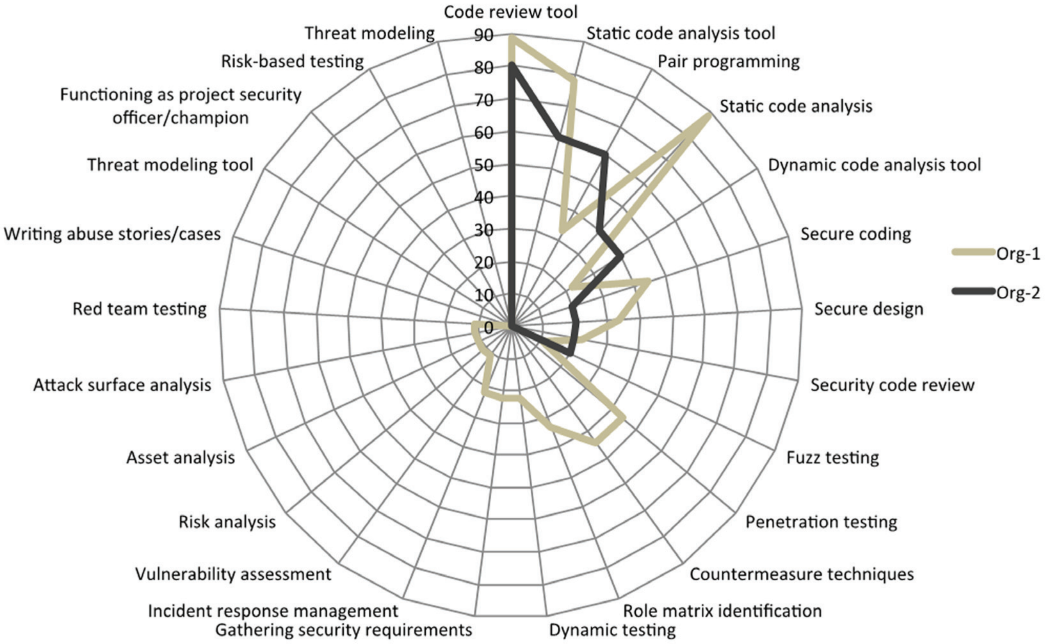


Figure 6. Comparison of usage level among testers between the 2 organisations

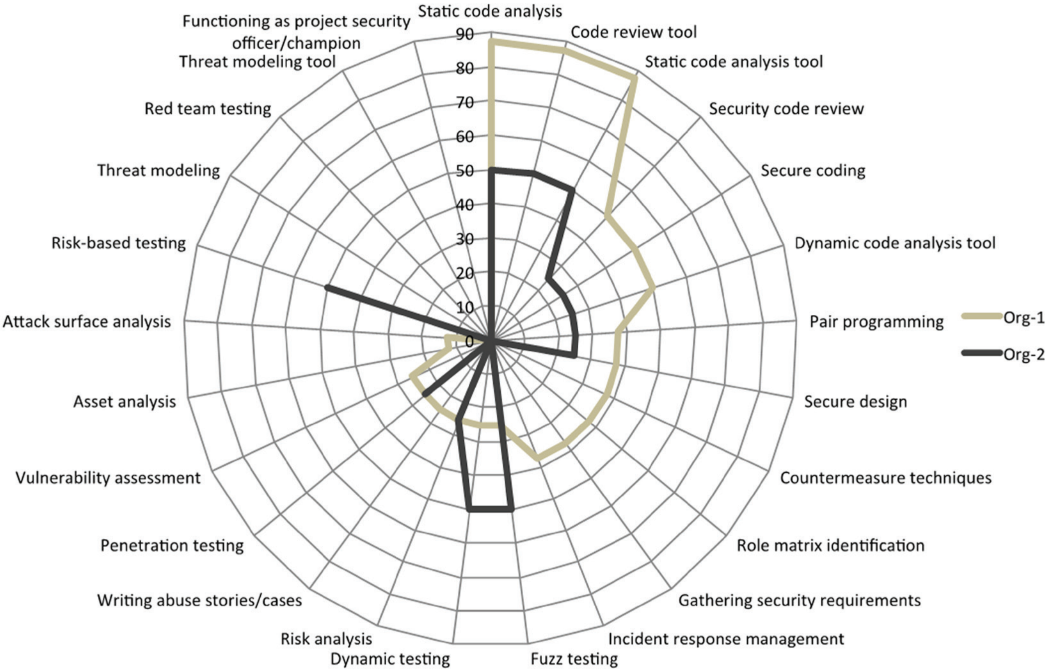


Table 10. Usage (Average %) of software security activities grouped by categories

	Developers		Architects		Testers	
	Org-1	Org-2	Org-1	Org-2	Org-1	Org-2
Frequently Used Activities	57	62	72	60	75	44
Secure Design and Coding	38	19	39	20	44	25
Threat Modeling and Risk Management	13	11	16	0	18	4
Security Testing	19	12	18	10	25	28
Requirements and Release	19	6	15	0	33	0

testing approaches used during high level testing such as risk-based testing. This could well explain the reason for the dissimilarity among testers in both organizations.

In a similar way, to explain the similarity among architects in both organizations, we found that 91% of architects in Org-1 are developers while all of the architects in Org-2 (100%) are developers. It is thus safe to conclude that developers are relatively similar in their skill levels across both organizations.

4.3. RQ3: Which Training Needs are Important to the Organizations?

In Org-1, secure design, secure coding, and penetration testing are the top-3 activities where the 3 roles expressed training needs (see Figure 8). In Org-2, secure design, risk analysis, and secure coding are the top-3 training needs expressed by the 3 groups. Secure coding and dynamic code analysis tool are in the top 3 for developers. Dynamic testing (Black box testing) and attack surface analysis

Figure 7. % of developers with moderate - expert skill levels compared between the 2 organisations

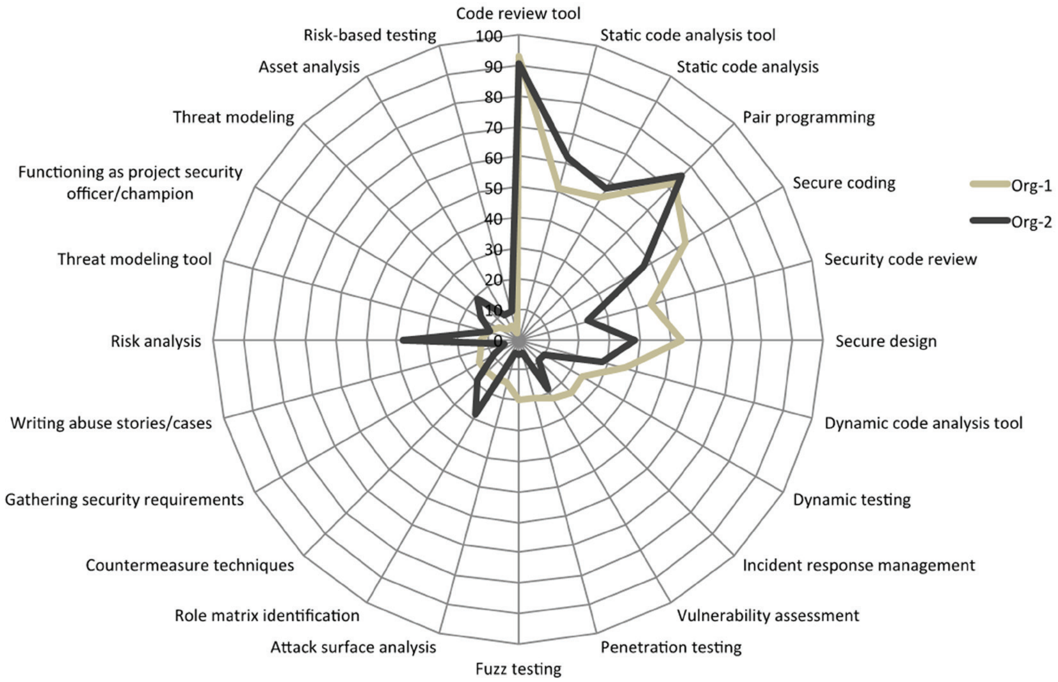
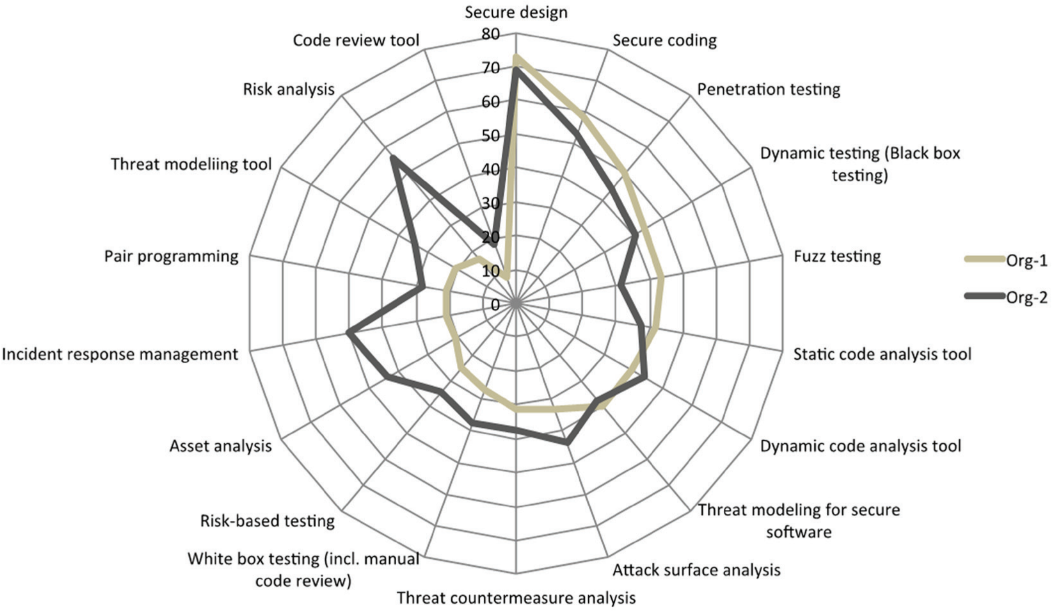


Figure 8. % of Training Needs across all roles compared between the 2 organizations



are in the top 3 for architects and lastly, asset analysis and incident response management are in the top 3 for testers.

The strongest statistically significant correlation with respect to training needs is found in the developer group (0.68). The similarity between the training needs for the architect group is weak (0.45). There is no similarity in the training needs among testers in both organizations. There are some similarities in the expressed training needs among developers irrespective of the organization. For instance, secure design, secure coding and dynamic testing are common areas of training needs for teams in both organizations.

In conclusion, secure design is indicated as the single most important training need expressed by teams in both organizations. There is thus a need to focus on how to address and assist agile teams in the area of secure design. Architectural-related challenges such as lack of time, motivation to consider design choices, and unknown domain and untried solutions have been shown to affect agile development teams (Babar, 2009).

4.4. RQ4: How are Security Experience and the Perceived Need for Software Security Training Influenced by Years of Developer Experience?

Results (Table 10) show that training needs are similar across years of development experience in org-1. However, in org-2, those with more than 5 years of development experience express the need for more training significantly more than those with fewer years of experience. In org-1, the top-3 training needs for both groups (0-5 years and >= 6years) are secure design (84.6 vs. 70.6), Penetration testing (53.9 vs. 50), and secure coding (46.2 vs. 64.7). In org-2, about 70% and more of those with >= 6 years of experience show training needs in secure design, risk analysis and secure coding. Whereas only 37.5% and less of those with fewer years of experience express training needs in all areas.

Based on our observations and interactions with both organizations, the similarities and differences in training needs between experienced and less experienced groups in both organizations may be explained by the differences in the organizations' cultures. Org-1 has a culture where teams learn from and interact with each other. They integrate new employees in their process through on-the-job training and introduction to job functions. We believe this culture could create similar training needs perceptions irrespective of years of development experience. However, in Org-2, there appears to be less synergy across teams. Creating on-the-job training and learning from each other are areas where Org-2 is currently improving.

In addition, 20% of those with 0-5 years in Org-1 have security experience which is in contrast with none in Org-2. Since this security experience is present in both groups in Org-1, it could create similar training needs perceptions for security activities.

We can thus infer that training needs may or may not be influenced by years of development experience. Factors such as an organization's working culture, teams' distribution, teams' interactions, security experience, and how new employees are integrated could be responsible for training needs perceptions across different years of experience (Table 11).

Table 12 lists the ratio of respondents with (Yes) and without (No) security experience within each years of development category. In both organizations, more respondents with many years of development experience indicate to have security experience. Evidence from the analysis shows that most young graduates have not had software security education and training, as the "0-5" year range has the lowest proportion of respondents with security experience in both organizations.

Zhu et al. (2013) argued that only a small fraction of developers are well trained in secure software development. This is because most Computer Science (CS) and Software Engineering (SE) curricula train students in programming and application development but not secure software development. As a result, CS and SE graduates are not trained on programming techniques to reduce security bugs and vulnerabilities and would unintentionally introduce avoidable security bugs in the application. While this result is not surprising, we believe it should be a call to integrate software security education in the curriculum for the next generation of CS and SE graduates.

Table 11. Test of difference between years of experience and training needs

	Mean		Median		Wilcoxon
	0-5	>=6	0-5	>=6	
Org-1	33.76	35.29	38.46	33.82	0.962
Org-2	17.36	52.52	12.50	50.00	8.282e-07

Table 12. Security experience vs. years of experience

	Proportion			
	Org-1		Org-2	
Years of development	Yes	No	Yes	No
0-5	0.2	0.8	0.00	1.00
6-10	0.5	0.5	0.17	0.83
>10	0.88	0.12	0.55	0.45
Pearson Correlation		-1		-1
P-value		<2.2e-16		<2.2e-16

4.5. RQ5: What is the Relationship Between Usage of, and Skill in Software Security Activities?

Correlation analysis between indicated skill levels and usage of activities show that skill drives usage of activities. In both organizations, the correlation result is very high at more than 0.9 and statistically significant at 95% confidence interval. Regardless of the cost of activity, we found that teams do well in activities where they indicate high level of skills. The studies by Baca & Carlsson (2011) and Ayaew et al. (2013) report code review to be detrimental in cost and benefit and pair programming to have marginal benefit and detrimental in cost to agile. However, our findings reveal that code review and pair programming are well practiced in both organizations and are areas where respondents indicate high skill levels.

Pair programming is an important practice in eXtreme Programming (XP) and by itself includes the art of code review (Beck, 1999). In addition, peer code review is claimed to catch about 60% of the defects (Boehm & Basili, 2005). These could explain the reasons both organizations have adopted these practices. The work of Dybå et al. (2004) that investigated the factors affecting software developer acceptance and utilization of Electronic Process Guides (EPG) corroborates this finding. Their results suggest that software developers are mainly concerned about the usefulness of the EPG regardless of whether it is easy to use, how much support they receive, or how much they are influenced by others.

On the other hand, we could hypothesize that management can increase usage in certain software security activity if they invest into increasing the team's skill in this area.

4.6. Discussions and Implications

A brief summary of our research questions and results is presented in Table 13.

Through interviews we discovered that certain security relevant tools (e.g. static analysis tools) are not used for finding security defects. This implies that simply making tools available will not improve security, unless the tools are actually used with security in mind.

Although both organizations deliver solutions for critical infrastructures, Org-1 has a higher level of security awareness, which is driven by the security expert group. This context is important in order to understand why this organization's usage is higher than the other. We need to further investigate the drivers for increase in software security adoption in an organization, such as research efforts, government funding and policies, education, and commitments by management to security.

Furthermore, the results from this survey show gaps in secure software development and opportunity for improvement. Among the development team, secure coding is practiced by less than half of the developers in both organizations. Invariably, over 50% of the developers are not paying attention to secure coding. The main question is whether this number is an acceptable risk for the management. Similarly, secure design is practiced by less than 40% of architects in both organizations. The high level of individual and team autonomy in agile settings requires a careful balance with respect to software security integration. While different approaches to integrate software security into agile teams have been proposed (Baca et al., 2015; Bartsch, 2011; ben Othmane et al., 2014), there are still many challenges about how to achieve it. The cost and benefit in terms of additional activity such as in ben Othmane et al. (2014) and additional security personnel, as in Baca et al. (2015) need to be acceptable to the agile team and management.

An important result from this survey is that secure design is the highest training need expressed by all roles in both organizations. We believe that this is not accidental. The need for secure design is corroborated in Arce et al. (2014). Critics of agile software development have argued that the lack of attention to design and architectural issues is a serious limitation of the agile approach (Dybå & Dingsøyr, 2008; Rosenberg & Stephens, 2003). About 60% of defects in a system is introduced during design (Bernstein & Yuhas, 2005), and fixing defects after release is a hundred times costlier than fixing it during requirement or design (Boehm & Basili, 2005). In terms of security defects in design, the strongest statement comes from a group of software security professionals (Arce et al., 2014): While a system may always have implementation defects, we have found that the security of

Table 13. Summary of results per research question

RQs	Conclusion
RQ1: Which software security activities are most used within the organizations?	Use of code review tool, static code analysis, and pair programming
RQ2: Which skills are common to both organizations?	Use of code review tool, pair programming, and static code analysis
RQ3: Which training needs are important to the organizations?	The organizations agree on secure design and secure coding, and additionally they identify training need in penetration testing and risk analysis
RQ4: How are security experience and the perceived need for software security training influenced by years of developer of experience?	Security experience increases with development experience, but perceived need for software security varies between organizations
RQ5: What is the relationship between usage of, and skill in software security activities?	Usage increases for activities where teams have a high level of skill

many systems is breached due to design flaws. In agile development, the lack of a complete overview of the system leaves room for unidentified risks during design.

Clearly, there is a need for more practice-oriented research efforts to find an acceptable approach that can help agile organization move towards their "adequate" level of security. We argue that security loopholes could be created by any team or individual within the organization with weak approaches to security. There are two major points to ponder in this result regarding software security adoption: 1) How can skill be increased in specific software security areas relevant to the development team and the goal of the organization? and 2) How can we create an environment that make replication of software security successes possible among teams? Creating a learning environment is central to point 1. Although agile development and learning are highly related (Aniche & de Azevedo Silveira, 2011), building a learning environment for security is not that easy. Differences in technologies and team autonomy are just two of the challenges to consider.

5. CONCLUSION

We have investigated the current usage, team competencies and training needs in software security activities among two agile organizations. We found that both organizations are similar in using certain activities such as use of code review tool, pair programming, and static code analysis/tool. These activities may or may not be used specifically for security. In core security activities such as threat modelling, secure design and coding, the two organizations are different. One performs more activities than the other due to a higher level of awareness created by the security expert group. Furthermore, skill drives the usage of activities. Secure design is consistently expressed by all roles in both organizations as the top most area where there is a need for training.

We identify learning and knowledge transfer as important to increase software security usage among teams. However, it requires that an enabling environment be built, as software security is unlikely to happen without a driver.

ACKNOWLEDGMENT

The work in this paper was supported by the Research Council of Norway through the project SoS-Agile: Science of Security in Agile Software Development (247678). We are grateful to our industrial partners and the survey respondents.

REFERENCES

Adams, E. (2012). The Biggest Information Security Mistakes that Organizations Make and How to Avoid Making Them. Retrieved from https://web.securityinnovation.com/the-biggest-information-security-mistakes-that-organizations-make

Allen, J. (2005). Governing for enterprise security (CMU/SEI-2005-TN-023). Retrieved from http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=7453

Aniche, M. F., & de Azevedo Silveira, G. (2011). Increasing learning in an agile environment: Lessons learned in an agile team. *Paper presented at the Agile Conference (AGILE).*

Arce, I., Clark-Fisher, K., Daswani, N., DelGrosso, J., Dhillon, D., Kern, C., . . . West, J. (2014). Avoiding The Top 10 Software Security Design Flaws. Retrieved from https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf

Ayalew, T., Kidane, T., & Carlsson, B. (2013). Identification and Evaluation of Security Activities in Agile Projects In Secure IT Systems (pp. 139–153). Springer.

Babar, M. A. (2009, September 14-17). An exploratory study of architectural practices and challenges in using agile software development approaches. *Paper presented at the 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture.* doi:10.1109/WICSA.2009.5290794

Baca, D., Boldt, M., Carlsson, B., & Jacobsson, A. (2015). A Novel Security-Enhanced Agile Software Development Process Applied in an Industrial Setting. *Paper presented at the 2015 10th International Conference on Availability, Reliability and Security (ARES)*. doi:10.1109/ARES.2015.45

Baca, D., & Carlsson, B. (2011). Agile development with security engineering activities. *Paper presented at the 2011 International Conference on Software and Systems Process*.

Bartsch, S. (2011). Practitioners' perspectives on security in agile development. *Paper presented at the 2011 Sixth International Conference on Availability, Reliability and Security (ARES)*. doi:10.1109/ARES.2011.82

Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70–77. doi:10.1109/2.796139

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

Bernstein, L., & Yuhas, C. M. (2005). *Trustworthy systems through quantitative software engineering* (Vol. 1). John Wiley & Sons. doi:10.1002/0471750336

Beznosov, K., & Kruchten, P. (2004). Towards agile security assurance. *Paper presented at the 2004 workshop on New security paradigms*.

Boehm, B., & Basili, V. R. (2005). *Software defect reduction top 10 list Foundations of empirical software engineering: the legacy of Victor R* (Vol. 426). Basili. doi:10.1007/3-540-27662-9

de Win, B., Scandariato, R., Buyens, K., Grégoire, J., & Joosen, W. (2009). On the secure software development process: CLASP, SDL and Touchpoints compared. *Information and Software Technology*, *51*(7), 1152–1171. doi:10.1016/j.infsof.2008.01.010

Dybå, T. (2000). An instrument for measuring the key factors of success in software process improvement. *Empirical Software Engineering*, *5*(4), 357–390. doi:10.1023/A:1009800404137

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50*(9), 833–859. doi:10.1016/j.infsof.2008.01.006

Dybå, T., Moe, N. B., & Mikkelsen, E. M. (2004). An empirical investigation on factors affecting software developer acceptance and utilization of electronic process guides. *Paper presented at the 10th International Symposium on Software Metrics*. doi:10.1109/METRIC.2004.1357905

Fong, E., & Okun, V. (2007). Web Application Scanners: Definitions and Functions. *Paper presented at the 40th Annual Hawaii International Conference on System Sciences*. doi:10.1109/HICSS.2007.611

Greenwood, D. J., & Levin, M. (2006). *Introduction to action research: Social research for social change*. SAGE publications.

Howard, M., & Lipner, S. (2006). *The Security Development Lifecycle*. Microsoft Press.

ISO/IEC. (2009). Information technology -- Security techniques -- Evaluation criteria for IT security -- Part 1: Introduction and general model: ISO/IEC 15408-1:2009.

Jaatun, M. G. (2012). Hunting for Aardvarks: Can Software Security be Measured? In G. Quirchmayr, J. Basl, I. You, L. Xu, & E. Weippl (Eds.), *Multidisciplinary Research and Practice for Information Systems* (pp. 85–92). Springer Berlin Heidelberg. doi:10.1007/978-3-642-32498-7_7

Jaatun, M. G., Cruzes, D. S., Bernsmed, K., Tøndel, I. A., & Røstad, L. (2015). Software Security Maturity in Public Organisations. *Paper presented at the Information Security: 18th International Conference ISC 2015*, Trondheim, Norway. Doi: doi:10.1007/978-3-319-23318-5_7

Knapp, T. R. (1990). Treating ordinal scales as interval scales: An attempt to resolve the controversy. *Nursing Research*, *39*(2), 121–123. doi:10.1097/00006199-199003000-00019 PMID:2315066

McGraw, G. (2004). Software Security. *IEEE Security and Privacy*, *2*(2), 80–83. doi:10.1109/MSECP.2004.1281254

McGraw, G. (2005). *The 7 Touchpoints of Secure Software*. Dr.Dobb's Journal.

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

McGraw, G., Migues, S., & West, J. (2016). Building Security In Maturity Model (BSIMM 7).

Microsoft. (2012). Security Development Lifecycle for Agile Development. Retrieved from https://msdn.microsoft.com/en-us/library/windows/desktop/ee790621.aspx

OWASP. (2006). CLASP concepts. Retrieved from https://www.owasp.org/index.php/CLASP_Concepts

OWASP. (2016). Software Assurance Maturity Model. Retrieved from http://www.opensamm.org/

Oyetoyan, T. D., Cruzes, D. S., & Jaatun, M. G. (2016). An Empirical Study on the Relationship between Software Security Skills, Usage and Training Needs in Agile Settings. *Paper presented at the 2016 11th International Conference on Availability, Reliability and Security (ARES)*. doi:10.1109/ARES.2016.103

Payment Card Industry. (2016). Payment Card Industry (PCI) Data Security Standard - Requirements and Security Assessment Procedures: PCI DSS v3.2.

Robinson, H., & Sharp, H. (2004, June 6-10). Extreme Programming and Agile Processes in Software Engineering. *Proceedings of the 5th International Conference, XP 2004*, Garmisch-Partenkirchen, Germany. Doi: doi:10.1007/978-3-540-24853-8_16

Rosenberg, D., & Stephens, M. (2003). *Extreme programming refactored: the case against XP*. Apress.

Wäyrynen, J., Bodén, M., & Boström, G. (2004). *Security engineering and eXtreme programming: An impossible marriage? Extreme programming and agile methods-XP/Agile Universe 2004* (pp. 117–128). Springer. doi:10.1007/978-3-540-27777-4_12

Zhu, J., Lipford, H. R., & Chu, B. (2013). Interactive support for secure programming education. *Paper presented at the 44th ACM technical symposium on Computer science education*. doi:10.1145/2445196.2445396

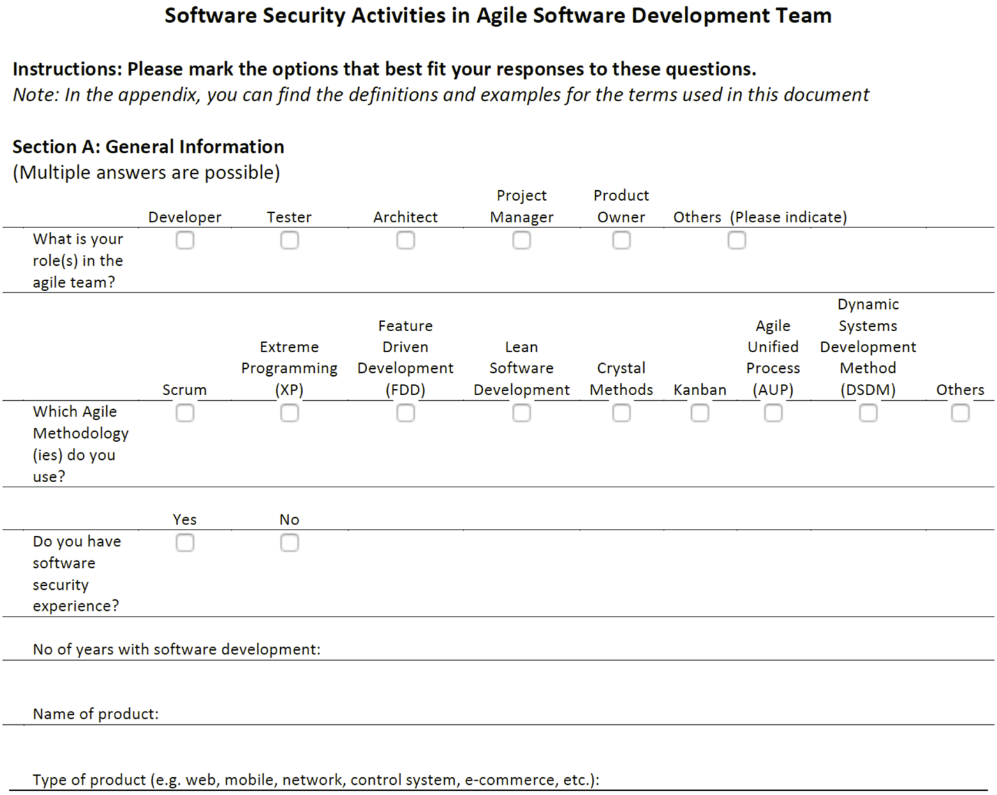
ENDNOTES

[1] https://www.r-project.org/

[2] https://samate.nist.gov/SARD/testsuite.php

APPENDIX A– SURVEY INSTRUMENT

Figure 9. Section A: General Information



Software Security Activities in Agile Software Development Team

Instructions: Please mark the options that best fit your responses to these questions.
Note: In the appendix, you can find the definitions and examples for the terms used in this document

Section A: General Information
(Multiple answers are possible)

	Developer	Tester	Architect	Project Manager	Product Owner	Others (Please indicate)
What is your role(s) in the agile team?	□	□	□	□	□	□

	Scrum	Extreme Programming (XP)	Feature Driven Development (FDD)	Lean Software Development	Crystal Methods	Kanban	Agile Unified Process (AUP)	Dynamic Systems Development Method (DSDM)	Others
Which Agile Methodology (ies) do you use?	□	□	□	□	□	□	□	□	□

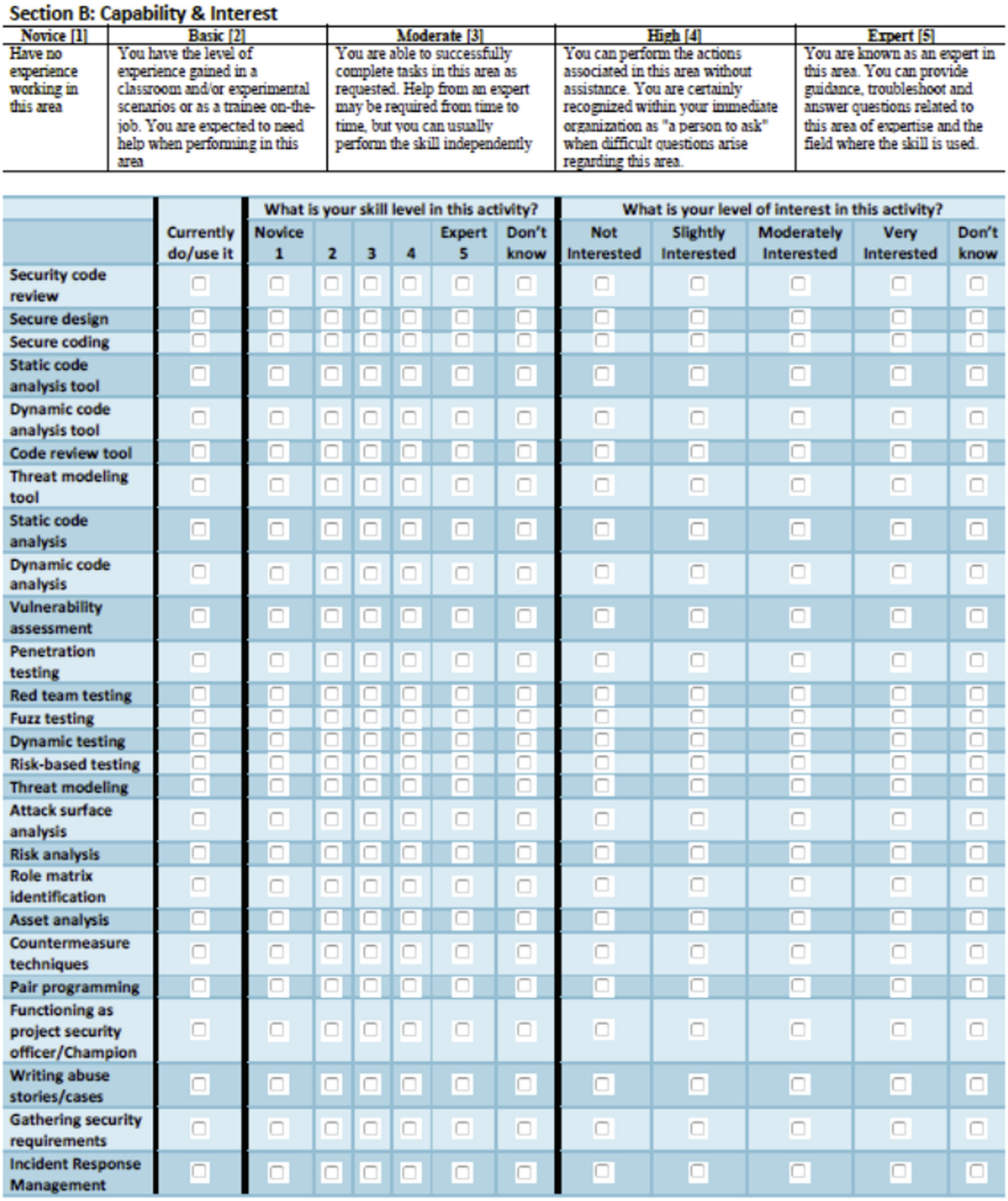
	Yes	No
Do you have software security experience?	□	□

No of years with software development:

Name of product:

Type of product (e.g. web, mobile, network, control system, e-commerce, etc.):

Figure 10. Section B: Capability & Interest

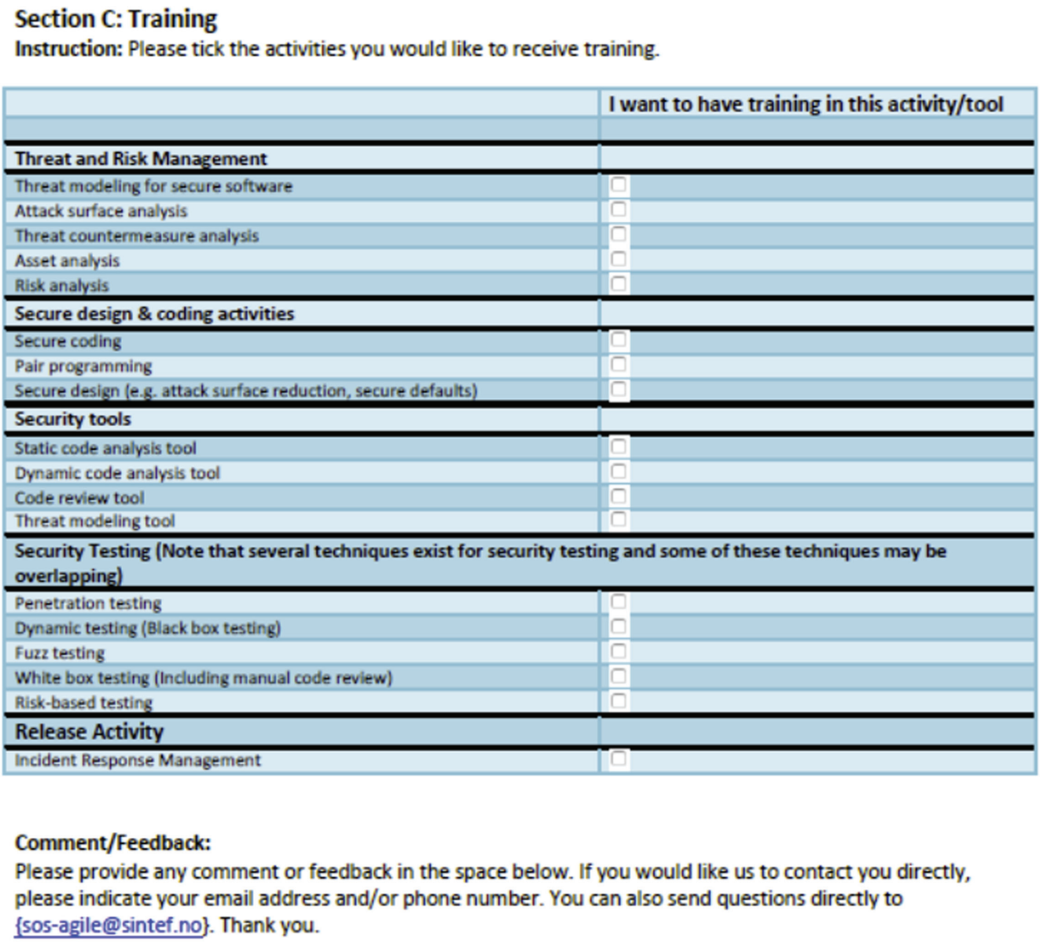


Section B: Capability & Interest

Novice [1]	Basic [2]	Moderate [3]	High [4]	Expert [5]
Have no experience working in this area	You have the level of experience gained in a classroom and/or experimental scenarios or as a trainee on-the-job. You are expected to need help when performing in this area	You are able to successfully complete tasks in this area as requested. Help from an expert may be required from time to time, but you can usually perform the skill independently	You can perform the actions associated in this area without assistance. You are certainly recognized within your immediate organization as "a person to ask" when difficult questions arise regarding this area.	You are known as an expert in this area. You can provide guidance, troubleshoot and answer questions related to this area of expertise and the field where the skill is used.

		What is your skill level in this activity?						What is your level of interest in this activity?				
	Currently do/use it	Novice 1	2	3	4	Expert 5	Don't know	Not Interested	Slightly Interested	Moderately Interested	Very Interested	Don't know
Security code review	□	□	□	□	□	□	□	□	□	□	□	□
Secure design	□	□	□	□	□	□	□	□	□	□	□	□
Secure coding	□	□	□	□	□	□	□	□	□	□	□	□
Static code analysis tool	□	□	□	□	□	□	□	□	□	□	□	□
Dynamic code analysis tool	□	□	□	□	□	□	□	□	□	□	□	□
Code review tool	□	□	□	□	□	□	□	□	□	□	□	□
Threat modeling tool	□	□	□	□	□	□	□	□	□	□	□	□
Static code analysis	□	□	□	□	□	□	□	□	□	□	□	□
Dynamic code analysis	□	□	□	□	□	□	□	□	□	□	□	□
Vulnerability assessment	□	□	□	□	□	□	□	□	□	□	□	□
Penetration testing	□	□	□	□	□	□	□	□	□	□	□	□
Red team testing	□	□	□	□	□	□	□	□	□	□	□	□
Fuzz testing	□	□	□	□	□	□	□	□	□	□	□	□
Dynamic testing	□	□	□	□	□	□	□	□	□	□	□	□
Risk-based testing	□	□	□	□	□	□	□	□	□	□	□	□
Threat modeling	□	□	□	□	□	□	□	□	□	□	□	□
Attack surface analysis	□	□	□	□	□	□	□	□	□	□	□	□
Risk analysis	□	□	□	□	□	□	□	□	□	□	□	□
Role matrix identification	□	□	□	□	□	□	□	□	□	□	□	□
Asset analysis	□	□	□	□	□	□	□	□	□	□	□	□
Countermeasure techniques	□	□	□	□	□	□	□	□	□	□	□	□
Pair programming	□	□	□	□	□	□	□	□	□	□	□	□
Functioning as project security officer/Champion	□	□	□	□	□	□	□	□	□	□	□	□
Writing abuse stories/cases	□	□	□	□	□	□	□	□	□	□	□	□
Gathering security requirements	□	□	□	□	□	□	□	□	□	□	□	□
Incident Response Management	□	□	□	□	□	□	□	□	□	□	□	□

Figure 11. Section C: Training



Section C: Training

Instruction: Please tick the activities you would like to receive training.

	I want to have training in this activity/tool
Threat and Risk Management	
Threat modeling for secure software	□
Attack surface analysis	□
Threat countermeasure analysis	□
Asset analysis	□
Risk analysis	□
Secure design & coding activities	
Secure coding	□
Pair programming	□
Secure design (e.g. attack surface reduction, secure defaults)	□
Security tools	
Static code analysis tool	□
Dynamic code analysis tool	□
Code review tool	□
Threat modeling tool	□
Security Testing (Note that several techniques exist for security testing and some of these techniques may be overlapping)	
Penetration testing	□
Dynamic testing (Black box testing)	□
Fuzz testing	□
White box testing (Including manual code review)	□
Risk-based testing	□
Release Activity	
Incident Response Management	□

Comment/Feedback:

Please provide any comment or feedback in the space below. If you would like us to contact you directly, please indicate your email address and/or phone number. You can also send questions directly to {sos-agile@sintef.no}. Thank you.

Tosin Daniel Oyetoyan is a post-doctoral fellow at SINTEF. He received his PhD from NTNU in 2015. He has previously worked as a senior software developer and analyst in the banking sector and has developed solutions for both healthcare and automation systems. His research interests are in software security, agile software development, software quality and maintenance, software design, code analysis, software testing, software refactoring, software metrics, and empirical software engineering. He is a certified secure software lifecycle professional (CSSLP) and a member of IEEE.

Martin Gilje Jaatun is a Senior Scientist at SINTEF. He graduated from the Norwegian Institute of Technology (NTH) in 1992, and received the Dr. Philos degree in critical infrastructure security from the University of Stavanger in 2015. Previous positions include scientist at the Norwegian Defence Research Establishment (FFI), and Senior Lecturer in information security at the Bodø Graduate School of Business. His research interests include software security, security in cloud computing and security of critical information infrastructures. He is vice chairman of the Cloud Computing Association (cloudcom.org), president of Cloud Security Alliance Norway, and a Senior Member of the IEEE.

Daniela S. Cruzes is a research scientist at SINTEF. Previously, she was adjunct associate professor at the Norwegian University of Science and Technology (NTNU). She worked as a researcher fellow at the University of Maryland and Fraunhofer Center for Experimental Software Engineering-Maryland. Dr. Cruzes is the project manager of the SoS-Agile (Science of Security for Agile Software Development) project funded by the Research Council of Norway. Her interests are agile software development, software security, global software engineering, empirical research methods, theory development and synthesis of software engineering studies.