Author version. Presented at 2016 11th International Conference on Availability, Reliability and Security (ARES)
Official version available at https://doi.org/10.1109/ARES.2016.103

An Empirical Study on the Relationship between Software Security Skills, Usage and Training needs in Agile Settings

Tosin Daniel Oyetoyan, Daniela Soares Cruzes, Martin Gilje Jaatun
Department of Software Engineering, Safety & Security
SINTEF ICT
Trondheim, Norway
{tosin.oyetoyan,daniela.s.cruzes,martin.g.jaatun}@sintef.no

***Abstract*—Organizations recognize that protecting their assets against attacks is an important business. However, achieving what is adequate security requires taking bold steps to address security practices within the organization. In the Agile software development world, security engineering process is unacceptable as it runs counter to the agile values. 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, we use survey to investigate software security usage, competence, and training needs in two agile organizations.

We find that (1) The two organizations perform differently in core software security activities but are similar when 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, Software security, Software security activities, Empirical study**

I. Introduction

Protecting the organization's assets from security threats is vital. Security is not an add-on functionality or product feature [21], and it is thus important that security is "built-in" in the process and the product. However, traditional security engineering process is often associated with additional development efforts and often invokes resentment among agile development teams [12], [10].

Some approaches have been proposed to integrate security activities into agile development, e.g. [23]. However, these approaches have been criticised to look similar to the traditional versions in terms of workload (e.g. performing a long list of security verification and validation tasks) [10]. 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 [18] and not the network. Thus, regardless of whether agile is incompatible with secure software development, the major discussion we should have is how to improve security within the agile context [8].

Previous studies [7], [5] 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; security requirements, role matrix, risk analysis, secure design principles, countermeasure graphs, coding rules, 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 [2], and what is currently practiced within several organizations. According to Allen [2], 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"*. This study is motivated based on the perceived gaps in software security by the management in the collaborated organizations. To address these gaps requires management to first understand the current status of software security practices and capability within their organization.

This study investigates existing practice, skills, and training needs within agile teams. We want to know 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) [22] has also being used to measure security practices in different organizations. Jaatun et al. [19] 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 [22]. However, it is heavyweight as a 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 Touchpoints [14]

TABLE II. 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

for this purpose.

The rest of this paper is organised as follows: We describe the background in Section II. We describe our case study context and study design in Section III. We present and discuss the results in Section IV. Finally we conclude in Section V.

II. BACKGROUND

A. Software Security Activities

We have drafted an instrument with questions on different software security activities (asterisked activities in Table I) from OWASP CLASP, Microsoft SDL for Agile, Common Criteria, and Cigital Touchpoints that have been used in previous studies [5], [7] (see Table I). The table also includes the additional activities "pair programming" and "countermeasure graph" considered in these studies; both are common security activities used in agile settings, e.g., when security experts rotate through programming pairs [8], [26].

The instrument has been jointly reviewed by a security professional, a security champion and a project manager in the two organizations. 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. For instance, whereas secure design involves many activities from "Threat modeling and risk management", we can argue that software designers could make assumptions about secure design when they include, e.g., authentication mechanisms [4]. However, performing a comprehensive threat analysis could reveal an insecure design, e.g., a possibility to bypass an authentication mechanism by directly navigating to an obscure webpage or resource. This could make it possible to spot such assumptions. 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 could lead to useful actions. These activities are divided into: Inception, threat modeling and risk management, secure design and coding, security tools, security testing, and release. Table II shows the software security activities. In addition, we provide a short explanation of each term we have used in the survey for the respondents.

III. STUDY DESIGN

A. Case study context

This study is carried out in 2 organizations(in the following referred to as "Org-1" and "Org-2"), that do business in telecommunication and transportation, respectively. Both organizations use agile practices (mostly Scrum and a mix of other agile flavors, e.g. XP) for their software development (see Table IV). The team set-up for 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 information security professional group, as they need to comply with various standards (e.g., PCI DSS compliance). The security group also coordinates security audit reports and follow up on reported security issues.

TABLE III. SOFTWARE SECURITY ACTIVITIES IN THE SURVEY

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

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.

B. Survey questions and hypotheses

We designed both an online questionnaire and a paper-based version. We 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.

1) Survey questions: The goal of the survey is to understand the state-of-practice, skill level and training needs of the agile teams regarding software security activities. As a result, we asked the following questions for the software activities listed in Table II:

Regarding skill levels among the developers, we use a 5-point Likert scale where 1=Novice, 2=Basic, 3=Average, 4=High, and 5=Expert. To avoid misunderstanding the scale values, we have provided a description for each value on the scale (see Table III for the description of each scale on how to choose skill level).

Q1 What is your skill level in this activity or tool?

TABLE I. SOFTWARE SECURITY ACTIVITIES FROM OWASP CLASP, MICROSFT SDL, CIGITAL TOUCHPOINTS (CT) AND COMMON CRITERIA (CC)

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)
	Repository Improvement (CC)	Critical Asset Analysis (CC)		Risk Based Testing (CT)*	Database Security Configuration (CLASP)
		UMLSec (CC)		Security Code Review (CLASP, SDL)*	
		Requirements Inspection (CC)		Security Testing (CLASP)	
		Countermeasure Graph (O)*			
		Cost Analysis (SDL)			
		Security Architecture (CLASP)			
		Secure Design Principles (CLASP)*			
		Identify Trust Boundary (CLASP)			

TABLE IV. 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

Regarding current usage, we ask a Yes/No question:

Q2 Do you currently use this activity or tool?

About training needs, we equally use a Yes/No question:

Q3 Do you want to have training in this activity or tool?

We are also interested to investigate the teams' software security experience directly and how long they have been involved with software development. To achieve this, we ask two additional questions:

Q4 Do you have security experience?
This is a Yes/No question - Nominal variable

Q5 Number of years with software development.
This is an interval variable

C. Investigations and Hypotheses

Undertanding the similarities and differences between organizations could help during replications and adoptions of software security activities and programs across different organizations. We identify four activities that are common to the two agile organizations. Use of code review tool, static code analysis, use of static code analysis tool, and pair programming. Pair programming is a major activity in Agile software development and by extension code review.

We make a distinction here; (1) Core security activities and (2) Activities that can be leveraged to deliver security. The above four activities are classified under (2), 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 the NIST SARD[1] 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. We grouped the four activities and named them "Frequently Used Activities" (FUA). This category name, FUA, is used to differentiate between both categories in the analysis.

1) Investigation about similarities and differences between the organizations: Our assumption is that developers have relatively similar skills in software security regardless of the organization where they currently work. We also assume that agile organizations have different usage patterns with software security activities. An agile team is mostly autonomous and self confident [24], 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 think that organizations would use activities that best fit their process and business needs. Therefore, we investigate whether the skills, usage and training needs in software security activities in both organizations are similar or different.

2) Hypotheses: We assume based on conventional wisdom that using an activity requires certain level of know-how. Therefore, teams would use activities where they have competence.

$H1$ Skill level is positively related to usage of security activities

We also assume that experienced developers would most probably have taken security related decisions during their development career, and thus have knowledge and experience in software security.

$H2$ Years of development experience is positively related to security experience

D. 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 [20] as an interval variable to statistically compare the skill levels between the organizations. In addition,

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

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. Finally, we 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.

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[2] for the analysis.

E. Assessment of reliability and validity

For this type of study, we are concerned about reliability, content validity, and construct validity [15]. 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.

1) Reliability: 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 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.

2) Content validity: 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 [14] (e.g. OWASP CLASP, Microsoft SDL, and Touchpoints). Thus content validity is built into the study from the beginning.

3) Construct validity: 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 III) 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.

4) External validity: 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 context 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.

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

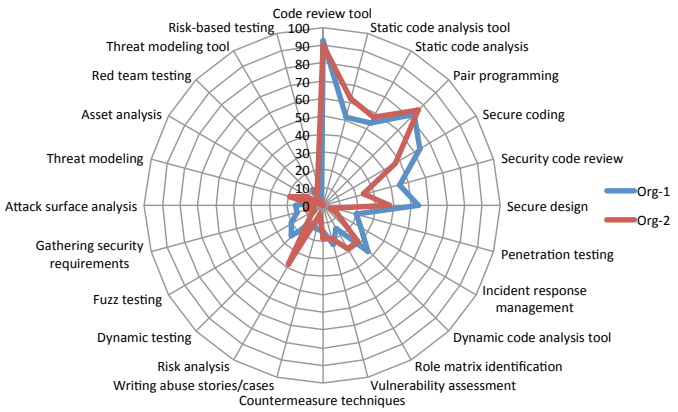

Fig. 1. % of developers with moderate - expert skill levels compared between the 2 organisations

IV. Results & Discussion

A. Results

We present the results of the survey and analysis conducted among the two organizations.

1) Skill (Q1): Here we report the activities where the majority (>= 50%) of respondents indicate to have moderate to expert skills. Among the developers in 'Org-1' (Figure 1), 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 the 2nd organization, 'Org-2' the situation is relatively similar. The activities/tools are in the order: 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.

2) Usage (Q2): We report activities where current usage is 40% or more (see Figures 2, 3, and 4). 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

TABLE V. CORRELATION ANALYSIS BETWEEN ORGANIZATIONS WRT SKILL, USAGE, AND TRAINING NEEDS GROUPED BY ROLES (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

TABLE VI. 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 & Coding	38	19	39	20	44	25
Threat Modeling & Risk Management	13	11	16	0	18	4
Security Testing	19	12	18	10	25	28
Requirements & Release	19	6	15	0	33	0

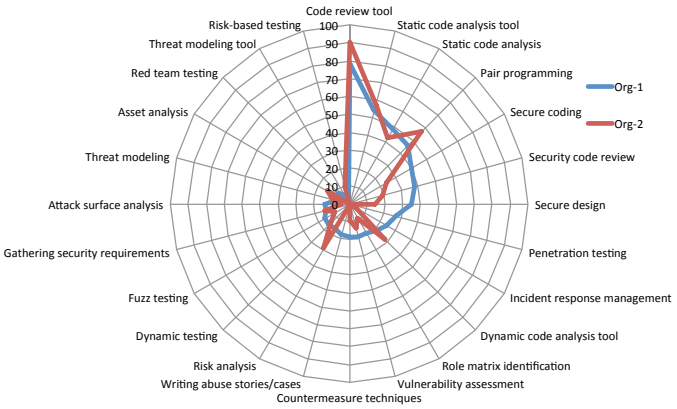

Fig. 2. Comparison of usage level among developers between the 2 organisations

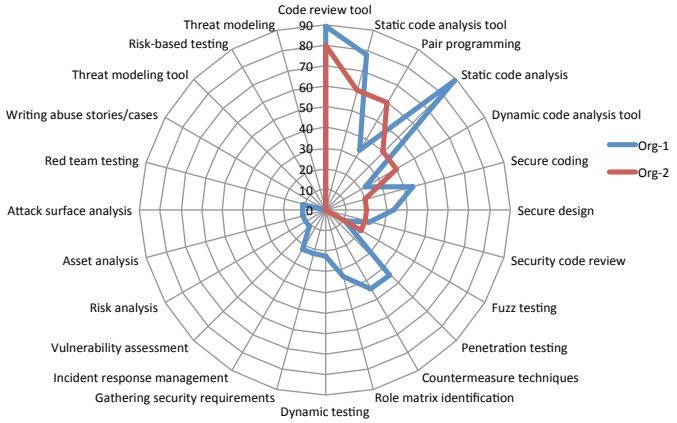

Fig. 3. Comparison of usage level among architects between the 2 organisations

when compared to research performed by Microsoft [1] 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, code review tool, static code analysis, secure coding, countermeasure techniques, and penetration testing are the most used activities. In Org-2, 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 compared to Org-2. We found no usage of such secure design activities among the architects.

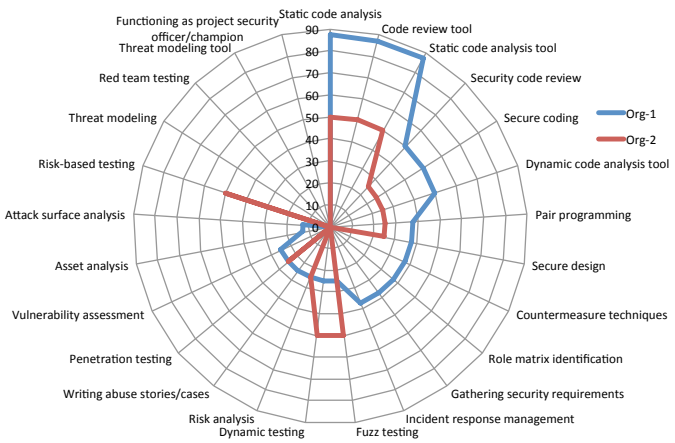

Fig. 4. Comparison of usage level among testers between the 2 organisations

At 40% usage threshold (see Figure 4), testers in Org-1 use code review tool, static code analysis, secure coding, dynamic code analysis tool, and security code review. While testers in Org-2 use 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.

3) Training Needs (Q3): In Org-1, secure design, secure coding, penetration testing, and use of dynamic code analysis tool are the top most activities where the 3 roles expressed training needs. In Org-2, secure design is the top most training

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

B. Comparison among Organizations

Table V lists the results of the correlation tests that compares the skill, usage and training needs between the 2 organizations.

a) Skills in software security activities: 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 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.

b) Usage of software security activities: The usage correlation results show that developers and architects are similar when we take 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 VI). 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.

TABLE VII. CORRELATION BETWEEN SKILL AND USAGE OF SOFTWARE SECURITY ACTIVITIES - H3

Skill vs. Usage (Q1 vs. Q2)		
	Pearson Correlation	P-value
Org-1	0.93	6.002e-11
Org-2	0.91	1.808e-10

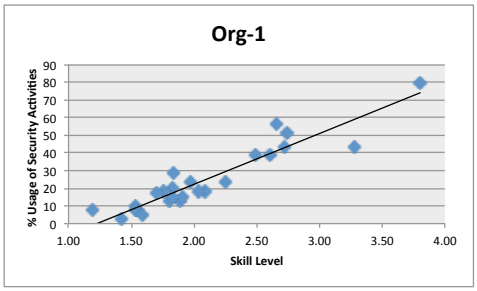

Fig. 5. Skill vs. Usage of Software Security Activities in Org-1

We interacted with the security director 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 share the same explanations in a) above. 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.

c) Training needs: 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 code analysis are areas of common training needs for teams in both organizations.

C. Results of Hypotheses

1) Skill vs. Usage (H1): 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 (see Table VIII and Figures 5 & 6). The studies by Baca & Carlsson [7] and Ayaew et al. [5] 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

Fig. 6. Skill vs. Usage of Software Security Activities in Org-2

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 [9]. In addition, peer code review is claimed to catch about 60% of the defects [13]. These could explain the reasons both organizations have adopted these practices. The work of Dybå et al. [17] 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 team's skill in this area.

2) Years of development vs. security experience (H2): Table VII lists the ratio of respondents with (Yes) and without (No) security experience within each years of development category. In both organization, 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. [27] 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.

D. Implications

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.

TABLE VIII. YEARS OF DEVELOPMENT EXPERIENCE VS. SECURITY EXPERIENCE (Q4 VS.Q5) - H4

	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	

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 [10], [6], [8], 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. [10] and additional security personnels, as in Baca et al. [6] 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. [4]. Critics of agile software development have argued that the lack of attention to design and achitectural issues is a serious limitation of the agile approach [16], [25]. About 60% of defects in a system is introduced during design [11], and fixing defects after release is 100x costlier than fixing it during requirement or design [13]. In terms of security defects in design, the strongest statement comes from a group of software security professionals [4]: *While a system may always have implementation defects, we have found that the security of 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 [3], building a learning environment for security is not that easy. Differences

in technologies and team autonomy are just few of the challenges to consider.

V. 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 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 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 may not happen without a driver.

Acknowledgements

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/O70). We are grateful to our industrial partners and the survey respondents.

References

[1] Ed Adams. The biggest information security mistakes that organizations make and how to avoid making them, 2012.

[2] Julia Allen. Governing for enterprise security. Technical report, DTIC Document, 2005.

[3] Mauricio Finavaro Aniche and Guilherme de Azevedo Silveira. Increasing learning in an agile environment: Lessons learned in an agile team. In *Agile Conference (AGILE), 2011*, pages 289–295. IEEE, 2011.

[4] Iván Arce, Kathleen Clark-Fisher, Neil Daswani, Jim DelGrosso, Danny Dhillon, Christoph Kern, Tadayoshi Kohno, Carl Landwehr, Gary McGraw, Brook Schoenfield, Margo Seltzer, Diomidis Spinellis, Izar Tarandach, and Jacob West. Avoiding the top 10 software security design flaws. Technical report, IEEE Computer Societys Center for Secure Design (CSD), 2014.

[5] Tigist Ayalew, Tigist Kidane, and Bengt Carlsson. Identification and evaluation of security activities in agile projects. In *Secure IT Systems*, pages 139–153. Springer, 2013.

[6] Dejan Baca, Martin Boldt, Bengt Carlsson, and Andreas Jacobsson. A novel security-enhanced agile software development process applied in an industrial setting. In *Availability, Reliability and Security (ARES), 2015 10th International Conference on*, pages 11–19. IEEE, 2015.

[7] Dejan Baca and Bengt Carlsson. Agile development with security engineering activities. In *Proceedings of the 2011 International Conference on Software and Systems Process*, pages 149–158. ACM, 2011.

[8] Steffen Bartsch. Practitioners' perspectives on security in agile development. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 479–484. IEEE, 2011.

[9] Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[10] Lotfi Ben Othmane, Pelin Angin, Harold Weffers, and Bharat Bhargava. Extending the agile development process to develop acceptably secure software. *IEEE Transactions on Dependable and Secure Computing*, 11(6):497–509, 2014.

[11] Lawrence Bernstein and Christine M Yuhas. *Trustworthy systems through quantitative software engineering*, volume 1. John Wiley & Sons, 2005.

[12] Konstantin Beznosov and Philippe Kruchten. Towards agile security assurance. In *Proceedings of the 2004 workshop on New security paradigms*, pages 47–54. ACM, 2004.

[13] Barry Boehm and Victor R Basili. Software defect reduction top 10 list. In *Foundations of empirical software engineering: the legacy of Victor R. Basili*, volume 426. 2005.

[14] Bart De Win, Riccardo Scandariato, Koen Buyens, Johan Grégoire, and Wouter Joosen. On the secure software development process: Clasp, sdl and touchpoints compared. *Information and software technology*, 51(7):1152–1171, 2009.

[15] Tore Dyba. An instrument for measuring the key factors of success in software process improvement. *Empirical software engineering*, 5(4):357–390, 2000.

[16] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9):833–859, 2008.

[17] Tore Dybå, Nils B Moe, and Edda M Mikkelsen. An empirical investigation on factors affecting software developer acceptance and utilization of electronic process guides. In *Software Metrics, 2004. Proceedings. 10th International Symposium on*, pages 220–231. IEEE, 2004.

[18] Elizabeth Fong and Vadim Okun. Web application scanners: Definitions and functions. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, HICSS '07, pages 280b–, Washington, DC, USA, 2007. IEEE Computer Society.

[19] Martin Gilje Jaatun, Daniela S. Cruzes, Karin Bernsmed, Inger Anne Tøndel, and Lillian Røstad. *Information Security: 18th International Conference, ISC 2015, Trondheim, Norway, September 9-11, 2015, Proceedings*, chapter Software Security Maturity in Public Organisations, pages 120–138. Springer International Publishing, Cham, 2015.

[20] Thomas R Knapp. Treating ordinal scales as interval scales: an attempt to resolve the controversy. *Nursing research*, 39(2):121–123, 1990.

[21] Gary McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

[22] Gary McGraw, Sammy Migues, and Jacob West. Building Security In Maturity Model (BSIMM 6), 2015. http://bsimm.com.

[23] Microsoft. SDL for Agile, 2016. https://www.microsoft.com/en-us/SDL/Discover/sdlagile.aspx [Online; accessed 30-April-2016].

[24] Hugh Robinson and Helen Sharp. *Extreme Programming and Agile Processes in Software Engineering: 5th International Conference, XP 2004, Garmisch-Partenkirchen, Germany, June 6-10, 2004. Proceedings*, chapter The Characteristics of XP Teams, pages 139–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[25] Doug Rosenberg and Matt Stephens. *Extreme programming refactored: the case against XP*. Apress, 2003.

[26] Jaana Wäyrynen, Marine Bodén, and Gustav Boström. Security engineering and extreme programming: An impossible marriage? In *Extreme programming and agile methods-XP/Agile Universe 2004*, pages 117–128. Springer, 2004.

[27] Jun Zhu, Heather Richter Lipford, and Bill Chu. Interactive support for secure programming education. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 687–692. ACM, 2013.