Exploring Security in Software Architecture and Design

Michael Felderer
University of Innsbruck, Austria

Riccardo Scandariato
Chalmers University of Technology, Sweden & University of Gothenburg, Sweden

A volume in the Advances in Information Security, Privacy, and Ethics (AISPE) Book Series

Published in the United States of America by
IGI Global
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA, USA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Names: Felderer, Michael, editor. | Scandariato, Riccardo, editor.
Title: Exploring security in software architecture and design / Michael Felderer and Riccardo Scandariato, editor.
Description: Hershey, PA : Information Science Reference, [2018] | Includes bibliographical references.
Identifiers: LCCN 2018008109| ISBN 9781522563136 (h/c) | ISBN 9781522563143 (eISBN)
Subjects: LCSH: Computer security. | Software architecture--Security measures. | Software engineering--Security measures.
Classification: LCC QA76.9.A25 E996 2018 | DDC 005.8--dc23 LC record available at https://lccn.loc.gov/2018008109

This book is published in the IGI Global book series Advances in Information Security, Privacy, and Ethics (AISPE) (ISSN: 1948-9730; eISSN: 1948-9749)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 11
Measuring Developers' Software Security Skills, Usage, and Training Needs

Tosin Daniel Oyetoyan
Western Norway University of Applied Sciences, Norway

Martin Gilje Jaatun
SINTEF Digital, Norway

Daniela Soares Cruzes
SINTEF Digital, Norway

ABSTRACT

Software security does not emerge fully formed by divine intervention in deserving software development organizations; it requires that developers have the required theoretical background and practical skills to enable them to write secure software, and that the software security activities are actually performed, not just documented procedures that sit gathering dust on a shelf. In this chapter, the authors present a survey instrument that can be used to investigate software security usage, competence, and training needs in agile organizations. They present results of using this instrument in two organizations. They find that regardless of cost or benefit, skill drives the kind of activities that are performed, and secure design may be the most important training need.

DOI: 10.4018/978-1-5225-6313-6.ch011

INTRODUCTION

Traditional security engineering processes are often associated with additional development efforts and are likely to be unpopular 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 too 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 better fits their process and practices. 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 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.

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*".

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" (Gary McGraw, 2006).

The studies by Ayalew et al. (2013), Baca and Carlsson (2011), and Morrison et al. (2017) have investigated security activities from cost and benefit dimensions to advise on frameworks and selection of security activities that can be integrated to agile software development. Jaatun et al. (2015) have used BSIMM to measure security practices but with focus on security maturity at an organisational level. Other studies not directly related to our work have looked into market skills relevant for cybersecurity jobs. For example, Potter and Vickers (2015) used a questionnaire to

answer and address the question of what skills does a security professional need in the current information technology environment, and they explored this question by looking at the current state of the Australian industry. Fontenele (Fontenele, 2017) developed a conceptual model and an ontological methodology to aid a robust discovery of the fittest expertise driven by the specific needs of cyber security projects, as well as benchmarking expertise shortages.

Our work differs from these studies as we have measured developers' skills and training needs along software security activities.

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.

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:

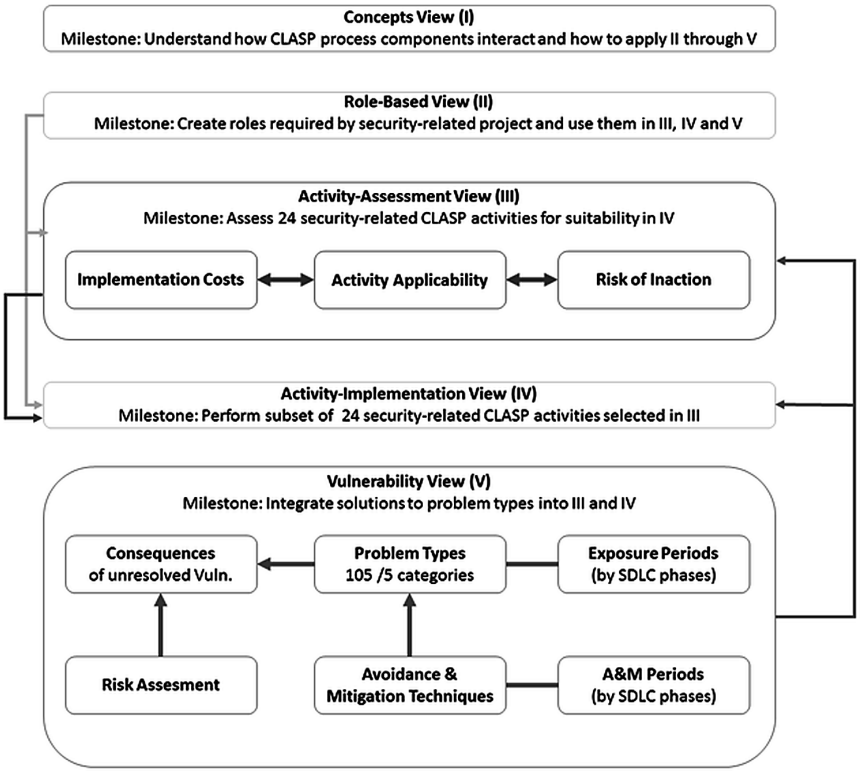
1. Institute awareness programs
2. Perform application assessments
3. Capture security requirements
4. Implement secure development practices
5. Build vulnerability remediation procedures
6. Define and monitor metrics
7. Publish operational security guidelines

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.

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);

Figure 1. CLASP Overview



- **"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 typically only need to be performed once at the beginning of the project.

Cigital Touchpoints

The Cigital Touchpoints (Gary McGraw, 2004; Gary McGraw, 2005) (de Win et al., 2009) 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.

Figure 2. The SDL-agile one-time and bucket requirements illustrated

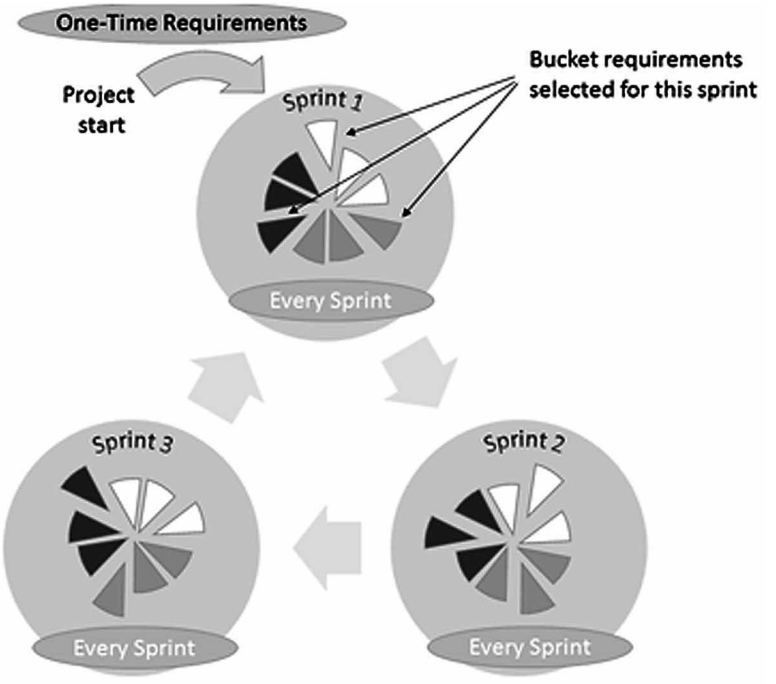
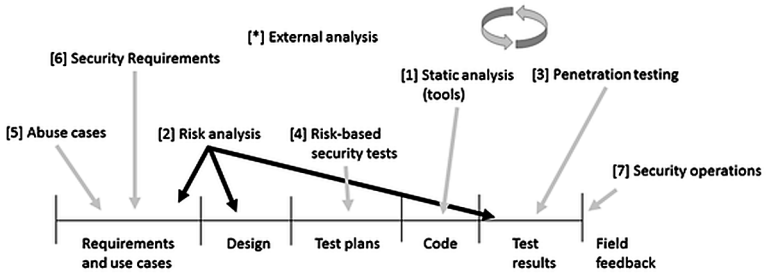


Table 1. MS SDL-agile activities – (o)ne-time, (s)print, and (b)ucket

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 3. The cigital touchpoints



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

ISO/IEC Application Security Standard

In 2011, the International Standards Organization published an application security standard as part of its 27000-series (ISO/IEC, 2011). We have not seen this standard in use in any of the organizations we have worked with, but it may prove relevant in the future.

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.

OpenSAMM

The OWASP 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.

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

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.

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 version of the BSIMM report (Gary McGraw et al., 2018) features results from 120 companies, measuring 116 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 116 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,

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)

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.

BSIMM 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., 2018). 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.

Common Criteria

The Common Criteria (ISO/IEC, 2009) (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. A security assurance requirement is intended to help achieve a certain level of confidence that the claimed (functional) security requirements are fulfilled, and typically relate to *how* the system is developed. 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.

The Top 10 Software Security Design Flaws

The IEEE Center for Secure Design has published a document (Arce et al., 2014) explaining how to avoid the ten most common software security design flaws. The recommendations are as follows:

1. Earn or Give, but Never Assume, Trust
2. Use an Authentication Mechanism that Cannot be Bypassed or Tampered With

3. Authorize after You Authenticate
4. Strictly Separate Data and Control Instructions, and Never Process Control Instructions Received from Untrusted Sources
5. Define an Approach that Ensures all Data are Explicitly Validated
6. Use Cryptography Correctly
7. Identify Sensitive Data and How They Should Be Handled
8. Always Consider the Users
9. Understand How Integrating External Components Changes Your Attack Surface
10. Be Flexible When Considering Future Changes to Objects and Actors

RESEARCH METHODOLOGY AND STUDY DESIGN

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. We used our survey instrument in a study carried out in 2 organizations (in the following referred to as "Org-1" and "Org-2"), that develop software in telecommunication and transportation, respectively. The case study is described in more detail in our previous work (Oyetoyan et al., 2016; Oyetoyan et al., 2017) investigating existing practice, skills, and training needs within agile teams. The survey instrument is intended to shed light 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 sections below describe the research questions, hypotheses, data collection procedure that we used in our case studies, the instruments used, and the type of data analysis performed.

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.

Our instrument is suitable for investigating whether the skills, usage and training needs in software security activities in several 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 that could be addressed include:

- Which software security activities are most used within the organization?
- Which training needs are important to the organization?
- How are security experience and the perceived need for software security training influenced by years of developer of experience?
- What is the relationship between usage of, and skill in software security activities?

Data Collection

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.

In addition, for the interpretation and discussion of the results, answers from the survey should be 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, should be compared with the collected information about the organizational contexts and documents.

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 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). 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 coding, security tools, security testing, and release. Table 5 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 4; the respondents were instructed to use this scale when assessing their own skill level.

For the software activities listed in Table 5, 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 have 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. In our experience, administering the questionnaire manually to the development teams on site will increase the response rate, and provide the opportunity to clarify questions that respondents might have.

The final questionnaire can be found in the original paper (Oyetoyan et al., 2017), and is also provided in Appendix A in this chapter. The skills are listed in Table 5, and additional explanations are further provided in Appendix A.

Comparing with the software security activities defined in BSIMM (Gary McGraw et al., 2018), we find that most of the activities in Table 5 are fully or partly covered by BSIMM, except "Countermeasure techniques", "Pair programming", and "Use of threat modelling tool". Threat modelling is equivalent to what BSIMM calls "Architecture Analysis", but this practice does not mention using a tool.

RESULTS

We used our survey instrument on two local companies (Oyetoyan et al., 2017), and in the following we briefly present some results of the survey and analysis conducted among the two organizations, discussing each research question in turn.

Table 4. 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 5. Mapping of software security activities

	CLASP	MS-SDL	CT	CC	Others	BSIMM Activities
Inception						
Functioning as project security officer/champion	*	*				SM1.2, SM2.3, T2.5, T2.7, T3.1, T3.5
Gathering security requirements	*	*	*	*		Partly covered by SR1.3 (Maybe whole practice SR)
Writing abuse stories/cases	*		*			AM2.1, ST3.5
Threat Modeling and Risk Management						
Threat modeling	*	*				Practice AM
Attack surface analysis	*	*				Partly covered by Practice AM
Countermeasure techniques	*	*				-
Asset analysis				*		Partly covered by AM1.2, CP2.1
Risk analysis	*		*	*		AA2.1
Role matrix identification	*					SM1.1
Secure Design and Coding						
Secure design	*	*	*			SFD1.2, SFD2.1, SFD2.2, SFD3.3
Secure coding	*	*	*			SR2.6, CR3.5
Pair programming					*	-
Static code analysis	*	*	*			Practice CR
Use of Security Tools						
Use of threat modeling tool					*	-
Use of dynamic code analysis tool					*	Partly covered by practice PT
Use of static code analysis tool					*	Partly covered by CR1.4
Use of code review tool					*	CR1.4, CR2.5, CR2.6, CR3.4
Security Testing						
Vulnerability assessment						Partly covered by practice AM
Penetration testing			*			Practice PT
Red team testing						PT1.1, PT1.3, PT3.1
Fuzz testing		*				ST2.6
Dynamic testing	*					Partly covered by Practice CR and ST
Risk-based testing			*			Practice ST
Security code review	*		*			Practice CR
Release						
Incident response management	*	*				CMVM1.1, CMVM2.1, CMVM3.3

Which Activities Are Most Used Within the Organizations?

Research performed by Microsoft (Adams, 2012) indicates that only 36% of developers are confident to write secure software. Our small sample indicates that this situation still persists. Our results show that the three most commonly used activities were:

- Use of code review tool
- Static code analysis
- Pair programming.

Note that none of these are necessarily pure software security activities, and may indeed be used without improving software security at all.

Which Training Needs Are Important to the Organizations?

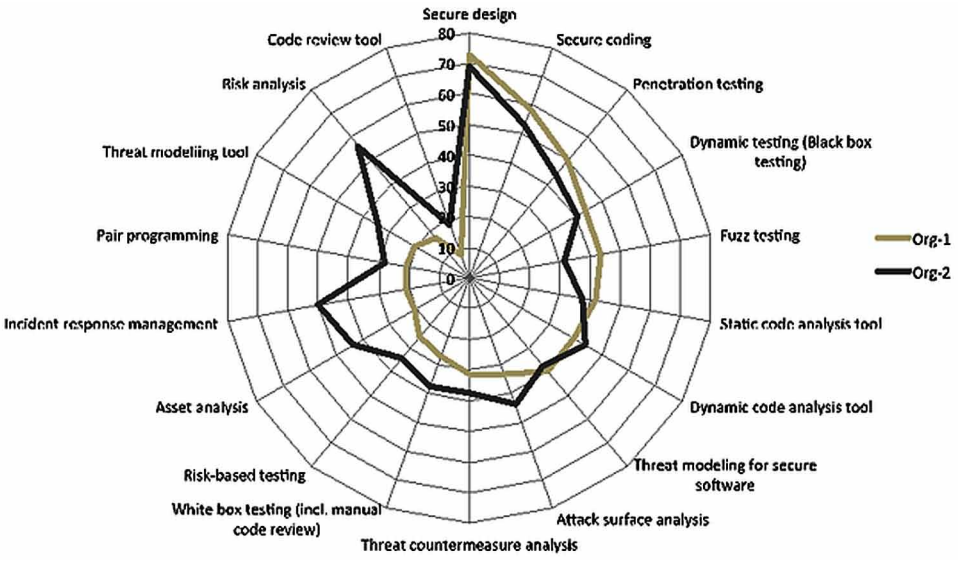
In our study, secure design was indicated as the single most important training need expressed by teams in both organizations. There is thus a need to focus on how to address and assist agile teams in the area of secure design. Architectural-related challenges such as lack of time, motivation to consider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

How Are Security Experience and the Perceived Need for Software Security Training Influenced by Years of Developer Experience?

We can 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.

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 in 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.

Figure 4. % of Training Needs across all roles compared between the 2 organizations



What Is the Relationship Between Usage of, and Skill in Software Security Activities?

Correlation analysis between indicated skill levels and usage of activities shows 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.

DISCUSSION

A brief summary of our research questions and results (Oyetoyan et al., 2017) is presented in Table 6. Note that despite the interpretation by Rindell et al. (Rindell et al., 2017), our contribution is not intended as another secure software development lifecycle.

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 our 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

Table 6. Summary of results per research question

RQs	Conclusion
1. Which software security activities are most used within the organizations?	Use of code review tool, static code analysis, and pair programming
2. 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
3. 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
4. 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

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 our 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 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.

Our impression is that none of the top 10 security design flaws (Arce et al., 2014) are particularly well known among developers, but many fall into the trap of equating authentication mechanisms with software security. Thus, this aspect is often implicitly covered, when good-practice standard authentication solutions are employed.

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.

CONCLUSION

We have presented an instrument for measuring the current usage, team competencies and training needs in software security activities in agile organizations. Our survey instrument complements maturity models such as BSIMM and OpenSAMM by focusing on the individuals rather than on organizations.

We have found that the individuals in our small sample of organizations were similar in terms of employing certain activities such as use of code review tool, pair programming, and use of static code analysis/tool, but since these activities may or may not be used specifically for security, particular focus on software security is necessary for these to have an impact on software security. Furthermore, skill drives the usage of activities, and we found that secure design may be the topmost area where there is a need for training.

We have identified learning and knowledge transfer as important to increase software security usage among teams.

ACKNOWLEDGMENT

The work in this chapter was supported by the Research Council of Norway through the project SoS-Agile: Science of Security in Agile Software Development (grant number 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), 2011. 10.1109/AGILE.2011.13

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). *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. 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 Availability, Reliability and Security (ARES), 2015 10th International Conference on. 10.1109/ARES.2015.45

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

Bartsch, S. (2011). *Practitioners' perspectives on security in agile development*. Paper presented at the Availability, Reliability and Security (ARES), 2011 Sixth International Conference on. 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. *Proceedings of the 2004 workshop on New security paradigms*.

Boehm, B., & Basili, V. R. (2005). *Software defect reduction top 10 list. In 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., & 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. *Software Metrics, 2004. Proceedings. 10th International Symposium on*. 10.1109/METRIC.2004.1357905

Fontenele, M. P. (2017). *Designing a method for discovering expertise in cyber security communities: an ontological approach*. University of Reading.

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.

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

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. 10.1007/978-3-319-23318-5_7

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 In. Addison-Wesley Professional.

McGraw, G., Migues, S., & West, J. (2018). *Building Security In Maturity Model (BSIMM 9)*. Academic Press.

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

Morrison, P., Smith, B. H., & Williams, L. (2017). *Surveying security practice adherence in software development*. Paper presented at the Hot Topics in Science of Security: Symposium and Bootcamp. 10.1145/3055305.3055312

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 Availability, Reliability and Security (ARES), 2016 11th International Conference on. 10.1109/ARES.2016.103

Oyetoyan, T. D., Jaatun, M. G., & Cruzes, D. S. (2017). A Lightweight Measurement of Software Security Skills, Usage and Training Needs in Agile Teams. *International Journal of Secure Software Engineering*, *8*(1), 27. doi:10.4018/IJSSE.2017010101

Potter, L. E., & Vickers, G. (2015). What skills do you need to work in cyber security?: A look at the Australian market. *Proceedings of the 2015 ACM SIGMIS Conference on Computers and People Research*. 10.1145/2751957.2751967

Rindell, K., Hyrynsalmi, S., & Leppänen, V. (2017). Busting a Myth: Review of Agile Security Engineering Methods. *Proceedings of the 12th International Conference on Availability, Reliability and Security*. 10.1145/3098954.3103170

Robinson, H., & Sharp, H. (2004). Extreme Programming and Agile Processes in Software Engineering. *5th International Conference, XP 2004 Proceedings*. 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? In 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. *Proceeding of the 44th ACM technical symposium on Computer science education*. 10.1145/2445196.2445396

APPENDIX

SURVEY INSTRUMENT AND EXPLANATION OF TERMS

Software Security Activities in Agile Software Development Team

Instructions: Please mark the options that best fit your responses to these questions.

Section A: General Information

(Multiple answers are possible, see Table 7)

Section B: Capability and Interest

See Tables 8 and 9.

Table 7.

	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 Method-ologies 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.):									

Table 8.

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.

Section C: Training

Instruction: Please tick the activities you would like to receive training on (see Table 10).

Comment/Feedback

Please provide any comment or feedback in the space below.

_____

Explanation of Terms in Questionnaire

See Table 11.

Table 9.

	Currently Do/Use It	What Is Your Skill Level in This Activity?						What Is Your Level Of Interest in This Activity?				
		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 modelling												
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												

Table 10.

	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	

Table 11.

I	Term	Definition	Examples
A	Abuse stories	Brief and informal stories that identify how attackers may abuse the system and jeopardize stakeholders' assets	
	Attack surface	All different points where an attacker could get into a system and get data out of the system	• user interface forms & fields • HTTP headers and cookies • APIs • Files • Databases • etc.
	Asset analysis	Identifying both physical and abstract assets of the organization. Assets are threat target. For example, an asset of an application might be a list of clients and their personal information; this is a physical asset. An abstract asset might be the reputation of an organization. Analysis may include identifying the trust levels (i.e. The level of access required to access the entry point is documented here)	
C	Code signing	Providing the stakeholder with a way to validate the origin and integrity of the system	
	Countermeasure	Action taken in order to protect an asset against threats	• Threat – Tampering with data • Countermeasures – appropriate authorization, hashes, digital signatures, etc.
D	Dynamic analysis tools	Automated runtime testing tools	• Penetration testing tools (e.g. ZAP, IBM AppScan, etc)
	Dynamic testing	Run-time verification of software programs	• memory corruption • user privilege issues • etc.
F	Final security review	A deliberate examination of all the security activities performed on a software application prior to release	
	Fuzz testing	Dynamic testing used to induce system failure by deliberately introducing malformed or random data to an application	
I	Incident response plan	A set of written instructions for detecting, responding to and limiting the effects of an information security event	
P	Pair programming	Two people create code where one writes the code while the other reviews each line of code as it is typed.	
	Penetration testing	Proactive and authorized attempt to evaluate the security of a system, by finding and exploiting vulnerabilities, technical flaws, or weaknesses to compromise the system	
Q	Quality gates/ bug bars	Minimum acceptable levels of security and privacy quality before the code goes into production	• All SQL statements must be parameterized before deployment • All API classes must be reviewed before deployment • Mandatory check for known vulnerabilities of all 3rd party libraries • All critical security bugs must be resolved
R	Red team testing	Simulate real-world attacks against an organization, challenging its defenses against electronic, physical and social exploits	Red team – an external[1] team with the goal to hack the system
	Risk analysis	An approach of gathering requisite data to make informed decision based on knowledge about asset, vulnerability, threat, impact, countermeasures and probability	
	Risk-based testing	Test approach that takes a risk into account by identifying and analyzing the risks related to the system	
	Role matrix	Identifying all possible user roles and their access levels to the system	

continued on following page

Table 11. Continued

I	Term	Definition	Examples
S	Secure coding	Development practices that assure secure software	• Input validation • parameterized SQL • etc.
	Secure design	Design practices that assure secure software	• reducing attack surface during design • placement of security checks before input processing • etc.
	Security code review	Manual review of source code for finding security bugs	
	Security metrics	Metrics that measure organization's defense against attacks	• Defect density • Windows of exposure (how long a security defect is open) • #Vulnerability • etc.
	Security patterns	A well understood solution to security problems	
	Security testing	An activity to assess a system for security bugs (technical flaws, vulnerabilities or weaknesses)	• Vulnerability assessment • Penetration testing • Dynamic testing (black box testing) • Code review (white box testing) • Automated analysis (dynamic and static)
	Static code analysis	Verification of source code	
	Static code analysis tools	Automated code review tools	• IDE vulnerability rule checker • Anti-XSS library • etc.
T	Threat modeling	An approach to identify, quantify, and address the security risks associated with a system	• identifying external dependencies • entry points • assets • trust levels • data flow diagrams • Categorize threats (attacker goals) e.g. Spoofing • Determine countermeasures (e.g. security controls) • etc.
U	UMLSec	Extension of Unified Modeling Language that allows to express security-relevant information within the diagrams in a system specification	
V	Vulnerability assessment	Scanning for security issues using a combination of automated tools and manual assessment techniques. The goal is to confirm the presence of a vulnerability without actually exploiting it	