How is Security Testing Done in Agile Teams? A Cross-Case Analysis of Four Software Teams

Daniela Soares Cruzes[1(✉)], Michael Felderer[2], Tosin Daniel Oyetoyan[1], Matthias Gander[2], and Irdin Pekaric[2]

[1] SINTEF Digital, Trondheim, Norway
{danielac,tosin.oyetoyan}@sintef.no
[2] University of Innsbruck, Innsbruck, Austria
{michael.felderer,matthias.gander,irdin.pekaric}@uibk.ac.at

Abstract. Security testing can broadly be described as (1) the testing of security requirements that concerns confidentiality, integrity, availability, authentication, authorization, nonrepudiation and (2) the testing of the software to validate how much it can withstand an attack. Agile testing involves immediately integrating changes into the main system, continuously testing all changes and updating test cases to be able to run a regression test at any time to verify that changes have not broken existing functionality. Software companies have a challenge to systematically apply security testing in their processes nowadays. There is a lack of guidelines in practice as well as empirical studies in real-world projects on agile security testing; industry in general needs a more systematic approach to security. The findings of this research are not surprising, but at the same time are alarming. The lack of knowledge on security by agile teams in general, the large dependency on incidental pen-testers, and the ignorance in static testing for security are indicators that security testing is highly under addressed and that more efforts should be addressed to security testing in agile teams.

Keywords: Security testing · Agile testing · Case study research

1 Introduction

Security testing can broadly be described as (1) the testing of security requirements that concerns confidentiality, integrity, availability, authentication, authorization, non-repudiation [16] and the testing to validate the ability of the software to withstand attack (resiliency) [28]. This process can be performed by showing conformance with the security properties, similar to requirements-based testing; or by trying to address known vulnerabilities, similar to traditional fault-based testing. It is essential to take testing into account in all phases of the secure software development lifecycle, i.e., analysis, design, development, deployment, as well as maintenance. Thus, security testing must be holistic covering the whole secure software development lifecycle. Proper security testing requires a mix of techniques as there is no single testing technique that can be performed to effectively cover all security testing and their application within testing

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 201–216, 2017.
DOI: 10.1007/978-3-319-57633-6_13

activities at unit, integration, and system level [2]. Nevertheless, many companies adopt only one security testing approach, for instance penetration testing.

Agile testing is one approach that is increasingly being adopted by software companies. This approach does not just mean testing on agile projects, but testing an application with a plan to learn about it and let the product information and customer feedback guide the testing. Agile testing involves immediately integrating changes into the main system, continuously testing all changes and updating test cases to be able to run a regression test at any time to verify that changes have not broken existing functionality [18, 23]. In agile software development, there is a focus on the feature implementation and delivery of value to the customer and, as such, non-functional aspects of a system should also be of attention. Non-functional requirements testing is challenging due its cross-functional aspects and lack of clarity of their needs by business in the most part of projects, therefore, although important, the non-functional requirements are often neglected in agile testing for many reasons, such as experience, culture, awareness, priority, cost and time pressure [5].

There is a lack of guidelines in practice as well as empirical studies in real-world projects on security testing; for agile projects in general needs a more systematic approach to security. The main contribution of this paper is to deepen relevant knowledge and experience on the characterization of security testing in an agile context. Based on the "traditional waterfall testing approaches and techniques", we have analyzed four teams and asked about how they perform these in the agile context. We then provide recommendations of ways to improve it based on lessons learned and good practices from the cases. In addition, we provide an improved understanding on how research and practice are aligned.

The remainder of the paper is organized as follows. In Sect. 2, we provide background on software and security testing. It also forms the backbone of the used interview guide. Section 3 presents the research methodology and describes how the studies were conducted. Section 4 presents the main findings of the case studies. Section 5 discusses the cross-case analysis findings. Finally, Sect. 6 concludes the paper and highlights directions of future work.

2 Background on Software and Security Testing

Software testing consists of all software development lifecycle activities, both static and dynamic, concerned with evaluation of software products and related artifacts to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects. Testing can be classified according to the three dimensions objective, scope, and accessibility shown in Fig. 1.

Test objectives are reason or purpose for designing and executing a test. The reason is either to check the functional behavior of the system or its nonfunctional properties. Functional testing is concerned with assessing the functional behavior of an SUT (System under Testing), whereas nonfunctional testing aims at assessing nonfunctional requirements with regard to quality characteristics like security or performance.

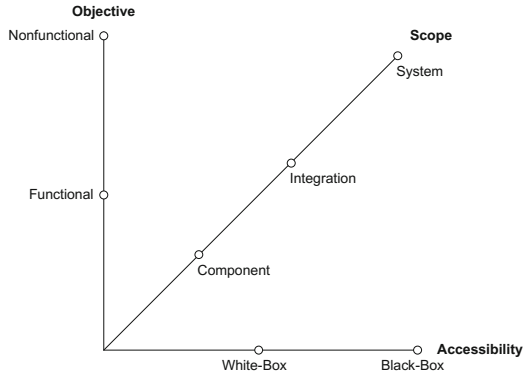

Fig. 1. Software testing dimensions objective, scope and accessibility (adopted from [16]).

The *test scope* describes the granularity of the SUT and can be classified into component, integration and system testing. It also determines the test basis, i.e., the artifacts to derive test cases. Component testing (also referred to as unit testing) checks the smallest testable component in isolation. Integration testing combines components with each other and tests those as a subsystem, that is, not yet a complete system. System testing checks the complete system, including all subsystems. A specific type of system testing is acceptance testing where it is checked whether a solution works for the user of a system. Regression testing is a selective retesting to verify that modifications have not caused side effects and that the SUT still complies with the specified requirements.

In terms of *accessibility* of test design artifacts we can classify testing methods into white-box and black-box testing. In white-box testing, test cases are derived based on information about how the software has been designed or coded. In black-box testing, test cases rely only on the input/output behavior of the software. This classification is especially relevant for security testing, as black-box testing, where no or only basic information about the system under test is provided, enables to mimic external attacks from hackers.

Security testing is testing of security requirements related to security properties like confidentiality, integrity, availability, authentication, authorization, and non-repudiation in addition to testing the resilience of the system against attack. In security testing, there are two principal approaches that can be distinguished, i.e., security functional testing and security vulnerability testing [33]. Security functional testing validates whether the specified security requirements are implemented correctly, both in terms of security properties and security mechanisms. Security vulnerability testing addresses the identification of unintended system vulnerabilities. It uses the simulation of attacks and other kinds of penetration testing attempting to compromise the security of a system by playing the role of a hacker trying to attack the system and exploit its vulnerabilities [1]. Furthermore, security vulnerability testing requires specific expertise, which makes it difficult and hard to automate [21]. By identifying risks in the system and creating tests driven by those risks, security vulnerability testing can focus on specific parts of a system implementation where an attack is likely to succeed.

Figure 2 abstracts from concrete security testing techniques mentioned before, and classifies them according to their test basis within the *secure software development life-cycle*, which takes security aspects into account in each phase of software development, i.e., analysis, design, implementation, deployment, maintenance, and additionally testing.

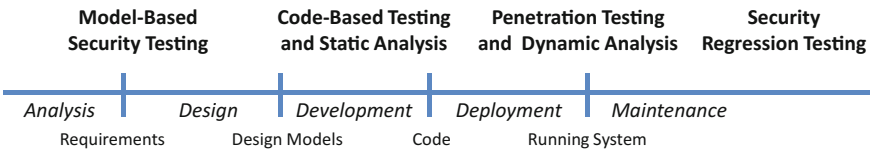

Fig. 2. Process for risk-based test strategy development (adopted from [16]).

Model-based security testing is grounded on requirements and design models created during the analysis and design phase. Examples are misuse cases and threat models. In misuse cases, test cases relating to an attacker's perspective are captured and used to exercise the system [31]. During the design, a threat model can be used to capture security issues and translated into test cases that can be used for security testing [20].

Code-based testing and static analysis is based on source, bytecode, or binary created during development. This testing approach in many cases uses static analysis tools to find code-based defects [6]. There is a range of issues that could be focused by a static analysis tool such as duplications, coding rules, code complexity, unit test coverage, and structural complexity. As regards security testing, specific frameworks exist that provide platform for common enumeration of security defects in the implementation and design. The Common Weakness Enumeration (CWE) [8] provides a formal list of software weaknesses. The OWASP Top-10 provides the list of the most common web application vulnerabilities [26]. The SANS Top-25 list shows the most widespread and critical errors that are applicable to all types of applications [11].

Penetration testing and dynamic analysis are based on running systems, either in a test or production environment. It is referred to as a black-box testing approach because the tester has no access to the source code of the system under test. Penetration testing seeks to break into running software but from ethical point of view. As a result, the rule of engagement must always be defined before such a test is carried out [28].

Refactoring and feature implementation may break existing security controls, increase the attack surface, and introduce new vulnerabilities into the system. In the agile context, it would be an activity that would need to be continuously performed to validate that the security properties of the system is not compromised.

2.1 Four Quadrants of Agile Testing

Crispin and Gregory [9] discuss the Agile Testing quadrants that are widely adopted in practice. Each quadrant in Fig. 3 reflects different reasons to test. Traditionally, software testing is involved late in the development process to detect failures, but typically not to prevent them. Companies focus almost exclusively on the right hand side (Q3 and

Q4), criticizing the product, but not playing a productive part in supporting the creation and guidance of the product (Q1 and Q2). In agile testing, the testers are not only involved in identifying, but also in preventing failures by continuous interaction with developers and customers. Automation is an important enabler for agile testing. Automation of the tests in Q1 is usually easiest to implement, and at the same time has a big impact on the process effectiveness. Tests in Q3 are usually performed manually. Tests in Q4 are heavily dependent on tools and specialized skill sets. But, manual exploratory testing by a knowledgeable security tester is indispensable to detect issues that automated tests can miss.

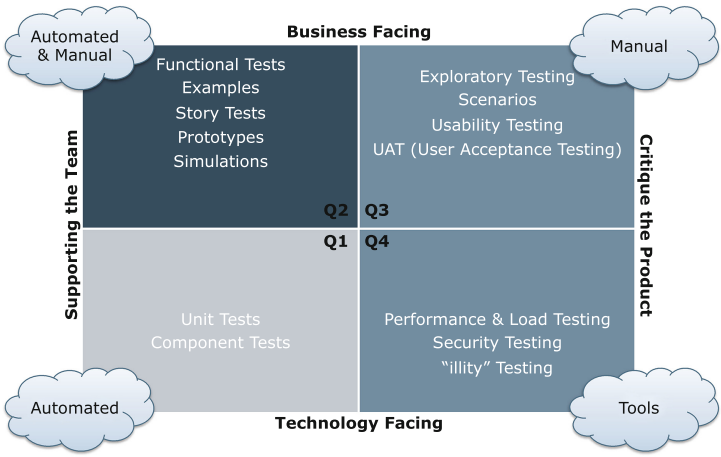

Fig. 3. Agile test quadrants [9]

Agile testing increases the need for improved communication and coordination between testers and developers, in addition to a new mind-set at the personal and organizational levels. In the rush to deliver functionality, most agile teams lack to think about security [5]. Authorization is often the only aspect of security testing that the agile teams consider as part of business functionality.

During the last years there have been several efforts to reconcile software security with the conflicting premises imposed by agile methodologies [4, 19, 24]. In a systematic review of agile challenges for secure software development Queslati et al. [24] conclude that the reported security assurance challenges are as follows: security assessment favors detailed documentation; tests are, in general, insufficient to ensure the implementation of security requirements; tests do not cover in general, all vulnerability cases; security tests are in general difficult to automate; and continuous changing of the development processes conflicts with audit needs of uniform stable processes.

Probably, the most widely known software security methodology is Microsoft's framework, which is integrated into the Microsoft Agile Security Development Lifecycle [22]. Other approaches also exist. Recently, Baca et al. [3] demonstrate how security features can be integrated into an agile software development method process at Ericsson AB. The approach focuses on risk management. Chólis et al. [7] describe a

case study of a software security testing process based on the Microsoft Software Development Lifecycle for Agile. The case company moves their software engineering teams from waterfall to agile. The case shows that a synchronization between the tasks of agile software engineering teams and the independent security team is possible. Türpe et al. [34] report on a one-year study of penetration testing and its aftermath at a major software vendor, and show how an agile development team managed to incorporate the test findings. Rindel et al. [30] describes a case of building a secure identity management system and its management processes. The project's steering group required the use of Scrum. In the implementations of this model the security testing, reviews and audits are viewed as normal stories in the sprint backlog and executed as part of the daily scrum.

Furthermore, security testing approaches for agile projects have especially been proposed for web applications [12, 32] and service-oriented systems [15]. These cases show how it is possible to integrate security testing into agile software development for specific system types. Our research comprises an independent study on the state of practice in security testing in agile teams.

3 Research Methodology

The overall goal of this paper is to investigate the role of security testing in agile teams, process-wise. For this purpose, we present the synthesis of the results of the four cases in security testing, highlighting the security engineering process, testing phases and techniques. The results of the interviews and context mapping provide insights into the recommended practices and lessons learned in the context of agile testing. The following three research questions (RQs) were investigated:

(**RQ 1**) How is the traditional security engineering process managed/organized in the agile teams?
(**RQ 2**) How does the agile teams perform security testing in each testing phase?
(**RQ 3**) How are traditional security testing techniques generally used in the agile software development lifecycle?

This study is carried out in four teams in two countries, i.e., Austria and Norway, within three organizations and denoted as 1, 2, 3-Team1, and 3-Team2, as shown in Table 2. Organizations 1 and 2 are located in the same country while organization 3 is located in another country. Organization 3 is a company with roughly 90 engineers. The team setup are both co-located and distributed. 3-Team1 has teams distributed in separate locations while 3-Team2 has the core development teams (frontend and backend) in the same location and interacts with a QA team that sits in a separate location. 3-team1 develops identity management APIs that are mainly consumed by other teams within the organization. They do not interact with external users. 3-Team2 on the other hand, develops solution for storage and processing of end user images and videos.

We prepared semi-structured interview guide (see Table 1) using a qualitative data collection approach that is based on in-depth literature review of the state-of-the-art in security testing. The interviews were compared with the collected information about the organizational contexts and interactions with the companies. The resulting interview

audios were then analyzed using the thematic analysis approach [10] to crosscheck and compare the answers in order to find behavioral confirmation and disconfirmation as well. The transcripts and recordings of the interviews were categorized, tabulated, and also analyzed by coding of the interviews. All the transcriptions and coding were validated with other researchers before analysis. By doing so, another researcher independently double-checked the codes and data to tag the key words, phrases and paragraphs. It is important to note that basic information on each context was considered (see Table 2). This information served as a context to better understand the points of view of each participant connected to the results. In this analysis, we considered in which areas the cases suggest the same points, where they differ, and where the cases conflict.

Table 1. Semi-structured interview guide

#	Questions	
1	Can you briefly describe the kind of system you develop? Back-end or Front-End?	
2	Can you give us a brief introduction of how your development team is organized? (Developers, Testers, Architects, CSOs, etc.), (Distributed, Co-located, etc.)	
3	How is your agile software development process? Which practices do you adopt? (Fill in the table with agile and lean practices)	
4	How is your security engineering process (for example, security requirements, secure design, secure coding, security testing) organized/managed in your team? Can you describe how you organize your security testing along these axes of the Fig. 1?	
5	Can you describe the kind of security testing that you perform in each testing phase listed below?	
	Phases of testing	**Components**
	Unit Testing	Classes, functions, statements, data
	Integration Testing	Modules, packages, etc.
	System Testing	System
	Regression Testing	Classes, Modules, System
	UAT Testing	System
	Production/Configuration Testing	System
6	Figure 2 shows the security testing techniques generally used in secure software development lifecycle. Could you talk about how you perform these activities in your agile software development? How often are security testing or security related activities done in your agile cycles? How do you decide when to perform them? How do you decide when not to perform them?	
7	Do you see benefits of performing security testing?	
8	On the test automation and continuous integration. Do you automate your testing activities? To what extent? How do you incorporate security testing in this process?	
9	Anything you would like to add?	

Table 2. Teams under study

Team	Team Size	Type of software	Other context information
1	20 Frontend and backend developers divided in teams of 5	Medical Information System	Applies a Scrum-based agile process; the software is certified according to medical standards
2	6 developers	Security service tools	Scrum-based agile process
3–A	21 developers (UI, Backend, Mobile, and Infrastructure)	Identity Management APIs that are consumed by other business units and teams	A mix of Agile Practices. Not specifically scrum by the book. DevOps approach is also spread used
3–B	22 developers (Frontend (web/mobile) and backend teams)	Mobile client and backend system for close storage and processing of images and videos	A mix of Agile Practices. Not specifically scrum by the book. DevOps approach is also spread used

4 Results

We collected our main findings in a mind map shown in Figs. 4, 5 and 6. These results are then discussed in more detail in the next subsections.

4.1 RQ 1: How Is the Traditional Security Engineering Process Managed/Organized in the Agile Teams?

We found three main themes from the interviews in relation to the roles and responsibility (Fig. 4). The first observation is that larger companies have their own chief security officer, who is not part of the teams to not interfere with any daily team activities. Sometimes the responsibility of the chief security officer overlaps with the project owner in order to ensure that the applications being developed do not impose security risks. One team mentioned that their project owner (PO) or project manager (PM) has domain-specific security knowledge, which is not the case for the other teams. In fact, for the smaller companies, there is no such chief security officer role. One problem that the teams experienced with involving the security officer is that it is hard to identify when to include him in the activities.

The second observation is that external experts are normally hired for penetration testing. However, a problem experienced by one of the companies is that external consultants do not have sufficient domain knowledge needed for security testing. Therefore, some domain-specific vulnerabilities are left undiscovered. The periodicity of the

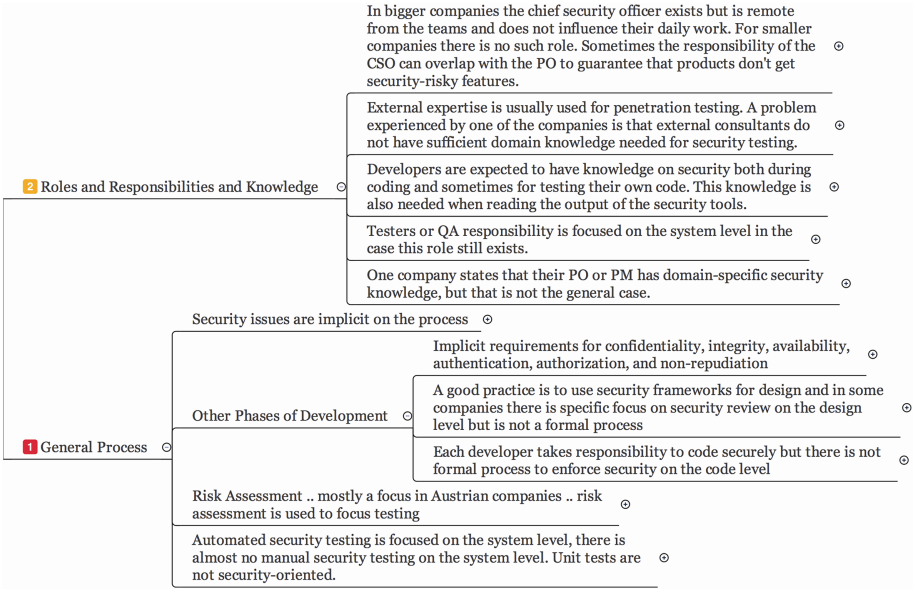

Fig. 4. Mind map: security software engineering process.

execution of these tests is quite ad-hoc, sometimes linked to big deliveries or when there are too many changes in the source code. The results of the tests are not completely integrated in the development process and almost never get into the planning of the activities of the sprint.

The third observation is that testers or QA personnel focus on the system level in the case this role still exists and the developers take care of the daily activities and developers are expected to have knowledge on security both during coding and sometimes for testing their own code. This knowledge is also needed when reading the output of the security tools. One interviewee said: "*We generally organize mainly as software developers, we generally have a software engineering role and we are expected to be with a broad knowledge, and skill set, computer science engineering and security and safe programming*". But there is no specific validation of this stated 'broad knowledge and skill set'. Another interviewee stated on some tool output: "*Normally, the errors are quite readable. From technician level, the developer that develops component should also understand the message of the tool. For instance, if the tool says, open API C# token found, hopefully developers also know what it says. The tools check very huge part, but they cannot check all. This is the responsibility that developer has while developing.*" It was clear that this knowledge was not something systematically evaluated or externalized, just assumed, as the agile mindset brings the focus to people instead of process and tools the teams are not completely sure of how much knowledge on secure coding was in the teams.

Automated unit testing is not security-oriented at all. Risk assessment is performed mostly by the Austrian teams (Team 1 and Team 2), and is applied to focus testing. One interviewer said: "*Yes, we are using risk assessment, it is a kind of matrix where we have*

on one hand probability occurrence and on the other hand importance of that stuff or if it can occur. We have this matrix and we are using it for small tools".

4.2 RQ 2: How Does the Agile Teams Perform Security Testing in Each Testing Phase?

To answer how security testing is performed in each testing phase, we analyzed the scope, objective and accessibility of the security testing, as shown in Fig. 5. With regard to the scope, unit tests are commonly used in agile teams, but typically not with a specific security focus. With some approaches for example testing positive and negative cases one team specifically mentions security focus for unit tests. Only one team highlights that security aspects are considered when negative unit tests, which are intended to fail, are executed.

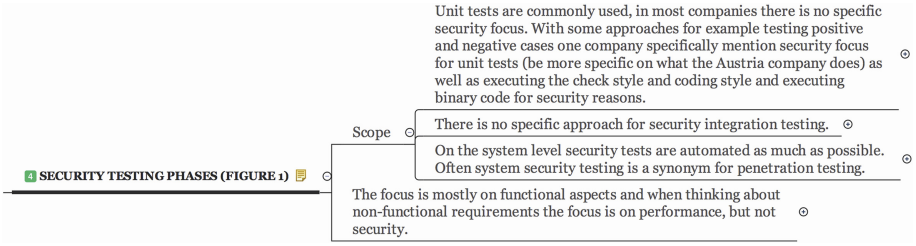

Fig. 5. Mind map security phases.

Static source and binary code analysis is performed for security reasons on the unit level. All teams stated that no specific security aspects are considered during integration testing. Security testing is most prominent on the system level. On this level security tests are typically a synonym for penetration testing, typically performed as black box testing. Security tests on the system level are to a large extent automated and there is almost no manual security testing on this level. White-box aspects are typically only considered during static source or binary code analysis.

When testing non-functional requirements, the focus in the interviewed teams is typically on performance. One interviewee said: "*We usually have unit test. And those are trying to exercise the happy path, which should already catch a many of basic the problems. We don't have much integration tests. We have also some performance tests. And that may go to the non-functional category, but we do not have much. We worry mostly on if the code works as it is supposed to work*".

4.3 RQ 3: How Are Traditional Security Testing Techniques Generally Used in the Agile Software Development Lifecycle?

For this question, the interviewees were asked to analyze Fig. 2. It shows the security testing techniques generally used in traditional secure software development lifecycle, i.e., model-based security testing, code-based testing and static analysis, penetration

testing and dynamic analysis as well as security regression testing. The interviewees were asked to talk about how they perform these activities in their agile software development and how often security testing or security related activities are done in their agile cycles. An overview of the results is shown in Fig. 6.

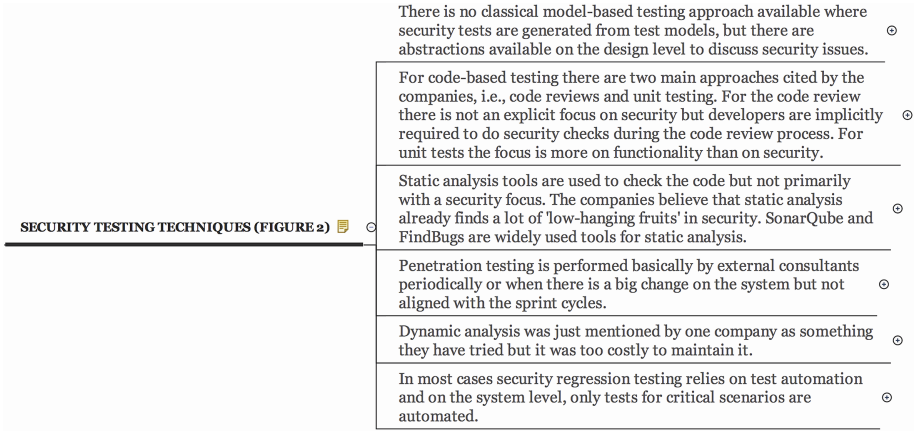

Fig. 6. Mind map security testing techniques findings.

In general, there is no classical model-based testing approach available where security tests are generated from test models, but there are abstractions available on the design level to discuss security issues. One interviewee said: *"We don't do any model-based testing. We consider security aspects as part of design and we don't try to buy a formal model around that. During development as we said we do code-based testing and static analysis. And that is probably on where most of our focus is. We have done some dynamic security tests in the past. As I said, those took a lot of manual effort and it was very unstable, it broke up often with some UI changes and it was hard to keep up".*

For code-based testing there are two main approaches referred to by the companies, i.e., code reviews and unit testing. When it comes to code review, there is no explicit emphasis on security, but developers are implicitly required to do security checks during the code review process. As for unit tests the focus is more on functionality than on security.

Static analysis tools are used to check the code but not primarily with a security focus. The interviewed teams believe that static analysis already finds the most important 'low-hanging fruits' in security. SonarQube and FindBugs are widely used tools for static analysis for the teams interviewed.

Penetration testing is performed basically by external consultants periodically or when there is a big change on the system but not aligned with the sprint cycles. One interviewee stated: "*We do penetration testing from external testers from the company, this was done together with the University of Innsbruck and plus our customers are doing against software. They are completely independent and we are not informed, we offer our aid only if there is a problem, and if we take Austrian medical network for*

example, it is not allowed to go live without testing from external company and that does not only involve our software but the whole system."

Dynamic analysis was only mentioned by one interviewee as something they have tried but it was too costly to maintain it. He said: "*It was taking too much time to keep it for us. And it requires a lot of manual integration and once that the scenario broke because of an UI change or something and then we would have again manual effort to fix that. For me, what makes code review and static analysis to work so well is that every time you compile the code you can see the feedback on it. On the dynamic tests, you cant do that very easily at that point you have to wait, and there is a lag between you writing your code and you receiving some feedback on it. Even if it is part of the development process, it doesn't happen right away. In my experience the further away from your commit, it less likely that you will either notice or be able to change it".*

In most cases security regression testing relies on test automation and on the system level only tests for critical scenarios are automated, but not a specific regression testing for security. One interviewee said: "*So what is working well is, I think our development processes are well structured and the biggest problem is, that we have frequent changes of user stories and that is very challenging on the one hand side on the development process and on the other side testing process. You have to adopt everything. The user stories are not from our customers, the problem the changing part is more about our c-level changes, on time this and one time that. So this is very big problem which also is very big problem for agile software development because it is very big problem".*

5 Discussion

Based on the results, we discuss recommendations for practice and research as well as limitations of this work.

5.1 Recommendations for Practice

With regard to the security engineering process, it is evident that the teams assume that developers have some security knowledge, but the issue is that they did not state how they conduct security engineering processes as well as what they need. For this reason, there is a demand for better use of guidelines for secure coding and testing practices like the OWASP guidelines [25]. Moreover, there should be a more systematic approach of spreading knowledge in security inside the teams. In a recent survey, Oyetoyan et al. [27] found that the developers' confidence in their software security knowledge is low, and therefore more efforts should be spend on getting the level of security knowledge higher at the companies. This is stronger in agile setting context because there is a strong dependency on people and not on process and tools. In addition code review and static analysis are used more and more in software projects, but without specific focus on security [27]. For this reason, processes of code reviews and static analysis should be more focused on security.

Even though the teams rely on penetration testing performed by externals, there is a danger of external penetration testers not having domain knowledge to catch important

vulnerabilities. While independent penetration testing is possible, there is a need that the penetration testing feedback is well integrated with the whole development process lifecycle [7, 34]. Chóliz et al. [7] have focused their study on the security testing activities, with the clear objective of synchronizing the tests from the independent security team with the agile rhythm of sprints, with frequent deliveries, of the software engineering teams, showing that the rate of found security vulnerabilities increased gradually. The results of Türpe et al. [34] suggest that penetration tests improve developers' security awareness, but long-lasting change of development practices is hampered if security is not properly reflected in the communicative and collaborative structures of the organization, e.g. by a dedicated stakeholder.

POs should have more security awareness because they are the only one responsible for maximizing the return on investment (ROI) of the development effort. In addition, the PO is responsible for product vision and constantly re-prioritizes the Product Backlog, thereby adjusting any long-term expectations such as release plans and making sure the team considers the stakeholders interests. The main issue with the explicit functional security requirements is that, most of the time stakeholders do not explicitly state them as requirements, and neither do the product owners. On the other hand, the non-functional security requirements are not features, which mean they never become a user story. In other words, they are not inserted into the product backlog. From the performed study, we see that security issues are implicitly handled on the process, but there is need for a more systematic approach to handle security issues in the development process. As shown by Rindel et al. [30] it is possible to have the security user stories as part of the product backlog.

5.2 Recommendations for Research

Research can help to increase knowledge and application of security testing in several respects. First, knowledge can be increased by the development of suitable courses and guidelines based on empirical evidence showing which approaches work in which context. Good efforts have been done in the last years [3, 7, 30, 34]. Therefore more empirical studies are needed which investigate challenges of security testing and derive respective evidence-based guidelines to address them.

With regard to model-based security testing, lightweight approaches are needed, which support the model creation, for instance, by learning of domain language concepts, based on design-level abstracts that are available also in agile teams. Also, a general understanding of the return of investment of model-based security testing approaches, which has already been highlighted as a challenge in [17], would help to apply such approaches efficiently. The issue of efficiently applying model-based testing approaches becomes even more critical when agile teams develop systems where the connection between safety and security is essential as in modern Internet-of-Things applications.

As seen in the results, system testing is often limited to penetration testing and testing of functional security requirements is often neglected. As automation is difficult to achieve fully, but at the same time, important for successful application in agile teams, suitable automation support and innovative techniques are required [29].

So far, security testing in agile teams makes little use of security risk assessments, which typically exist in an implicit or explicit for in other organization units. Risk assessment can be used to develop risk-based testing approaches [14], which can guide decisions during testing, and for instance help to select and prioritize security regression tests [13]. Baca et al. [3] shows that using a risk analysis approach, it s possible to find more severe risks, besides, more advanced skills and a deeper awareness of the problems become available. More research needs to be done in order to understand the best way to apply risk management in agile projects and especially on security.

5.3 Work Limitations

Common criticisms to a case study also apply to this study, among them one may list: uniqueness, difficulty to generalize the results, and the introduction of bias by participants and researchers. In our study, we generalized the findings from empirical statements to theoretical statements, which involved generalizing data from interviews and perceptions by discussing them in accordance with the literature. Interview data were though our primary source of information.

Qualitative findings are highly context and case-dependent. Our findings apply to software projects teams within four participating teams. However, all the participants were professionals using typical development technologies in a typical working environment, e.g., the natural setting demanded by the case study approach. We described the main characteristics of each case and company, including context and settings, data collection, analysis, and analysis process, as well as quotations with our major findings. This makes the results easier to generalize.

As commonly done in in-depth qualitative studies, we also had to do a trade-off between the number of participants, the duration and the cost of this study. The number of subjects interviewed in this context is not quantitatively significant, but gives deeper insights on the issues investigated in this work.

6 Conclusion

In this paper, we investigated by a cross-case analysis of four teams, two from Austria and two from Norway, how security testing is performed in agile teams. We investigated how the security engineering process is managed/organized in agile teams, how security testing is performed in each testing phase, and how security testing techniques are generally used in the secure software development lifecycle.

Although the study is based only on the results of a limited amount of agile teams, i.e., four, agile teams, we could derive recommendations for research and practice. The findings of this research are not surprising, but at the same time are alarming. The lack of knowledge on security by agile teams in general, the large dependency on incidental penetration testers, and the ignorance in static testing for security are indicators that security testing is highly under addressed and that more efforts should be addressed to security testing in agile teams.

In the future, we plan to replicate this study and to develop and evaluate suitable security testing approaches to support the adoption of security testing in agile teams through action research studies with industry.

Acknowledgments. This work was partially supported by the SoS-Agile (247678/070) project funded by the Research Council of Norway, and by MOBSTECO (FWF P 26194-N15) funded by the Austrian Science Fund. The authors are grateful to all involved in this study, specially the interviewees for their insights and cooperation and to the software companies for supporting this work.

References

1. Arkin, B., Stender, S., McGraw, G.: Software penetration testing. IEEE Secur. Priv. **3**(1), 84–87 (2005)
2. Austin, A., Williams, L.: One technique is not enough: a comparison of vulnerability discovery techniques. In: ESEM 2011, pp. 97–106 (2011)
3. Baca, D., Boldt, M., Carlsson B., Jacobsson, A.: A novel security-enhanced agile software development process applied in an industrial setting. In: ARES 2015, pp. 11–19 (2015)
4. Beznosov, K., Kruchten, P.: Towards agile security assurance. In: NSPW 2004, pp. 47–54 (2004)
5. Camacho, C.R., Marczak, S., Cruzes, D.S.: Agile team members perceptions on non-functional testing: influencing factors from an empirical study. In: ARES 2016, pp. 582–589 (2016)
6. Chess, B., McGraw, G.: Static analysis for security. IEEE Secur. Priv. **2**(6), 76–79 (2004)
7. Choliz, J., Vilas, J., Moreira, J.: Independent security testing on agile software development: a case study in a software company. In: ARES 2015, pp. 522–531 (2015)
8. Common Weakness Enumeration (CWE), 5 March, 2017. https://cwe.mitre.org/index.html
9. Crispin, L., Gregory, J.: Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley Professional, Boston (2009)
10. Cruzes, D., Dybå, T.: Recommended steps for thematic synthesis in software engineering. In: ESEM 2011, pp. 275–284 (2011)
11. CWE/SANS TOP 25 Most Dangerous Software Errors, 5 March 2017. https://www.sans.org/top25-software-errors/
12. Erdogan, G., Meland, P.H., Mathieson, D.: Security testing in agile web application development - a case study using the EAST methodology. In: Sillitti, A., Martin, A., Wang, X., Whitworth, E. (eds.) XP 2010. LNBIP, vol. 48, pp. 14–27. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13054-0_2
13. Felderer, M., Fourneret, E.: A systematic classification of security regression testing approaches. Int. J. Soft Tools Technol. Transf. **17**(3), 305–319 (2015)
14. Felderer, M., Schieferdecker, I.: A taxonomy of risk-based testing. Int. J. Softw. Tools Technol. Transf. **16**(5), 559–568 (2014)
15. Felderer, M., Agreiter, B., Breu, R., Armenteros, A.: Security Testing by Telling Test Stories. Modellierung **161**, 195–202 (2011)
16. Felderer, M., Büchler, M., Johns, M., Brucker, A.D., Breu, R., Pretschner, A.: Chapter one-security testing: a survey. Adv. Comput. **101**, 1–51 (2016)
17. Felderer, M., Zech, P., Breu, R., Büchler, M., Pretschner, A.: Model-based security testing: a taxonomy and systematic classification. Softw. Test. Verification Reliab. **26**(2), 119–148 (2016)

18. Fitzgerald, B., Stol, K.-J.: Continuous software engineering: a roadmap and agenda. JSS **123**, 176–189 (2017)
19. Keramati, H., Mirian-Hosseinabadi, S.: Integrating software development security activities with agile methodologies. In: AICCSA 2008 (2008)
20. Marback, A., Do, H., He, K., Kondamarri, S., Xu, D.: A threat model-based approach to security testing. Softw. Pract. Experience **43**(2), 241–258 (2013)
21. McGraw, G., Potter, B.: Software security testing. IEEE Secur. Priv. **2**(5), 81–85 (2004)
22. Microsoft, Agile Development Using Microsoft Security Development Lifecycle 5 March 2017. http://www.microsoft.com/en-us/sdl/discover/sdlagile.aspx
23. Moe, N.B., Cruzes, D., Dybå, T., Mikkelsen, E.M.: Continuous software testing in a globally distributed project. In: ICGSE 2015, pp. 130–134 (2015)
24. Oueslati, H., Rahman, M.M., Othmane, L., Ghani, I., Arbain, A.F.: Evaluation of the challenges of developing secure software using the agile approach. Int. J. Secure Softw. Eng. **7**, 17 (2016)
25. OWASP Foundation: OWASP Testing Guide v4. 5 March, 2017. https://www.owasp.org/index.php/OWASP_Testing_Project
26. OWASP Top 10. 5 March 2017. https://www.owasp.org/index.php/Top_10_2013-Top_10
27. Oyetoyan, T.D., Cruzes, D.S., Jaatun, M.G.: An empirical study on the relationship between software security skills, usage and training needs in agile settings. In: ARES 2016, pp. 548–555 (2016)
28. Paul, M.: Official (ISC)2 Guide to the CSSLP CBK, 2nd edn. (ISC)2 Press (2014)
29. Peischl, B., Felderer, M., Beer, A.: Testing security requirements with non-experts: approaches and empirical investigations. In: QRS 2016, pp. 254–261 (2016)
30. Rindell, K., Hyrynsalmi, S., Leppänen, V.: Case study of security development in an agile environment: building identity management for a government agency. In: ARES 2016, pp. 556–563 (2016)
31. Sindre, G., Opdahl, A.L.: Eliciting security requirements with misuse cases. Requirements Eng. **10**(1), 34–44 (2005)
32. Tappenden, A., et al.: Agile security testing of web-based systems via HTTP unit. In: Proceedings of Agile Conference. IEEE (2005)
33. Tian-yang, G., Yin-sheng, S., You-yuan, F.: Research on software security testing. World Acad. Sci. Eng. Technol. **70**, 647–651 (2010)
34. Türpe, S., Kocksch, L., Poller, A.: Penetration tests a turning point in security practices? In: Organizational Challenges and Implications in a Software Development Team, WSIW@SOUPS 2016 (2016)