International Journal of Secure Software Engineering

July-September 2010, Vol. 1, No. 3

Table of Contents

Special Issue on Software Security

Guest Editorial Preface

Research Articles

Agile Software Development:
The Straight and Narrow Path to Secure Software?

Torstein Nicolaysen, NTNU, Norway

Richard Sassoon, NTNU, Norway

Maria B. Line, SINTEF ICT, Norway

Martin Gilje Jaatun, SINTEF ICT, Norway

ABSTRACT

In this article, the authors contrast the results of a series of interviews with agile software development organizations with a case study of a distributed agile development effort, focusing on how information security is taken care of in an agile context. The interviews indicate that small and medium-sized agile software development organizations do not use any particular methodology to achieve security goals, even when their software is web-facing and potential targets of attack. This case study confirms that even in cases where security is an articulated requirement, and where security design is fed as input to the implementation team, there is no guarantee that the end result meets the security objectives. The authors contend that security must be built as an intrinsic software property and emphasize the need for security awareness throughout the whole software development lifecycle. This paper suggests two extensions to agile methodologies that may contribute to ensuring focus on security during the complete lifecycle.

Keywords: *Agile, Information Security, Scrum, Secure Software Engineering, Security*

1. INTRODUCTION

A decade or so ago, the waterfall model was the favored way of managing/building projects, resulting in a very formal approach where security was handled both implicitly and specifically. Due to the rigid and formal nature of the waterfall model, there was a place for security in specific parts of the process. This does not automatically mean that the waterfall model will make the software secure; it still requires skilled people and determination to create secure software.

Agile software development has become a buzzword, and most modern IT-companies brag about how they are using it. Scrum (Scrum Alliance, 2009) is a popular and widely used agile software development methodology, which contains no specific techniques or help for handling critical elements like security. As Scrum is more of a project management methodology, it might not be up to Scrum to

DOI: 10.4018/jsse.2010070105

handle all aspects of security, but it does define how the requirements are elicited and how to communicate with the customer. If done by the book, the customer has to request security and then prioritize it. If neither the customer nor the developers are concerned with security, it will most likely never end up in the product backlog, and therefore it will be neglected.

This article refers to software security as the resistance against misuse and/or attacks. Specific security features such as login functionality and encrypted communication are part of this, but even more important is *secure code* features, aiming at making the code unexploitable, preventing attacks like buffer overflow, XSS and similar.

The big question is how software security fits into software development projects where agile methodologies are used. Can agile methodologies be mixed with the rigid and formal processes associated with software security, and if so, how?

This article presents an empirical study of how agile software developers include security in their projects. It also presents a case study showing that software development without a persistent focus on security results in software with a number of vulnerabilities. Finally, the article presents two possible extensions to agile methodologies, intended to increase developers' awareness of software security.

2. BACKGROUND

Enabling information systems to communicate via open networks such as the Internet will always be associated with elements of risk. (Mavridis, Georgiadis, Pangalos, & Khair, 2001) correctly state that "Security risks cannot be entirely removed when transmitting information over the Internet". The European Parliamentary Technology Assessment (EPTA) network has made similar considerations and specifically expressed concerns that privacy is challenged by the increase in development of ICT applications for the healthcare sector (EPTA, 2006). Such concerns are also raised by others, such as (Ilioudis & Pangalos, 2001) and (van der Haak et al., 2003).

(Boström, Wäyrynen, Bodén, Beznosov, & Kruchten, 2006) detail an extension to the XP planning game that is intended to establish a balance between the conventional (document-centric and plan-driven) way of doing security engineering, and the iteration-centric, feedback-driven XP practices. This is relevant as they try to solve a problem closely related to ours. The main difference is that they are specific to the XP methodology and only try to integrate the security requirements engineering (software security) activity, whereas our approach is more generic for Agile methods and not focusing on just one specific security activity.

(Beznosov & Kruchten, 2004) attempt to find the pain points between agile methods and *security assurance*, and suggest some means on how to alleviate them. They group the problems and evaluate how good they match up against activities from security assurance. They focus on a specific problem, like Boström et al.'s approach, and do not seek to solve a more general problem.

(Siponen, Baskerville, & Kuivalainen, 2005) provide an example on how to integrate some security activities into agile development methods. They focus on four key security elements: security-relevant subjects, security-relevant objects, security classification of objects and subjects, and risk management. In the provided example where they apply their technique, it becomes apparent that it requires a lot more effort than what can be expected from an average developer. We therefore consider this too heavy for general applications with agile software development. Their result gives us an indication of what makes a process too thorough.

(Keramati & Mirian-Hosseinabadi, 2008) provide a semi-formal way of evaluating the agility of an agile method. When adding software security activities to an existing agile method, their work can be used to calculate how much the activity reduces the *degree of agility*. They also introduce a parameter named agility

reduction tolerance, which indicates how willing the organization is to accept heavyweight security activities about to be integrated with their agile methods.

3. EMPIRICAL STUDY

Six different software development companies in Trondheim, ranging from consulting firms, private enterprises and government-based organisations, were interviewed. The six companies were chosen due to their geographical location and their usage of agile methodologies. Each company was represented by a software developer, and half of them had some experience with software security. Only one of the companies had an extensive focus on software security, and many of their projects are subject to stricter security requirements than average. Examples include tax submission and insurance systems.

The interviews were carried out in order to confirm or falsify the following research hypothesis:

Software security is not a specific concern in an agile software development setting.

We expected the interviews to confirm that security is too often neglected during software development. However, we also hoped to uncover some new or existing techniques, as a "reality check" on how scientifically developed methods actually work in practice regarding how security can be ensured in agile projects.

In the following we present our findings from the interviews.

3.1. Functionality before Security

For most of the interviewed companies, functional requirements are more important than non-functional requirements. Only one of the companies had a somewhat clear method for software security. From a cynical business perspective, it is not difficult to understand why functionality is the priority, while security, performance, availability and the other quality attributes are neglected until they are needed.

It is difficult to calculate what is more cost-efficient when faced with the option between securing or not. Securing has a 100% certain price tag of <insert big number here>, while being hacked might be a calculated risk discounted down to <insert slightly smaller number here>. The accepted wisdom (Boehm & Basili, 2001) is that costs for fixing (security) bugs rise exponentially toward the end of the development process, so if the company is unwilling to focus on security early, it is not more likely to do it later. On the other hand, the company's reputation will be damaged, which is discussed in more detail later. This part is what makes it difficult to calculate the total cost. Money lost due to loss of customers might not be calculated into the initial risk analysis. Most of those interviewed were honestly concerned about this part, but some of the companies were in a dominant market position allowing them to take a punch from bad PR and loss of customers. This did not necessarily mean that they ignored security, but they did not have it as their top priority.

Running a business is about making money. Seen from a company's viewpoint, up-front securing a non-critical piece of software might not seem like a good return on investment right there and then. Functionality is what the customer ordered, and it is naturally the top priority, but it is still hard to comprehend that the non-functional requirements are so utterly neglected. Although the cases presented in the interviews might not seem like they needed security, there had rarely been conducted any formal security activities up front (or during the process for that matter), thus leaving a possibility for major security issues that never were considered.

3.2. Agile Versus Security

Of all the companies interviewed, only one had tried to combine software security and agile software development. The Norwegian Data Inspectorate enforces strict policies and

regulations when operating with sensitive data, which makes it hard to work agile. A lot of documentation has to be written. Each project usually needs a specific permission to access e.g. tax databases, involving routines for setting up servers, authorizations and many other activities that slow down agile methods. It would be unfortunate if every Norwegian citizen's tax details, social security number and personal information were available to all developers on a project. However, the friction between such routines and agile methods is noticeable. None of the interview subjects had any ideas as how to make that kind of processes more streamlined. This said; none of them gave any examples of how they had tried to. On the other hand, some of the interview subjects were eager to discuss what agile methods and software security had to do with each other.

3.3. Securing with Infrastructure

Many companies secure the software they create by protecting it with infrastructure like private networks, firewalls and restricted access to resources. This results in a situation where we end up with "crunchy shell around a soft, chewy center" (Cheswick, 1990), which might be viewed as an acceptable short-time solution in cases where time-to-market is critical. Not many of our interviewees had thought about what would happen if the system were taken outside the intended environment: If a new group of system administrators decided to move one of the servers outside the private intranet, data might be exposed to the public.

Some servers containing files only meant for internal use are completely open because they are on a private intranet. Some use real data sets for testing. They often contain semi-sensitive information like addresses, phone numbers and such. If the test-database is not given the same security focus as the rest of the system, it is probably an easy target for someone who wants that information. These are issues that infrastructure and routines cannot easily protect against. It is important to note that while a secure infrastructure is vital, it should always be in addition to software security.

3.4. Lack of Formal Knowledge about Software Security

Few of the interviewees had any formal knowledge about software security. This is understandable as it is a relatively new concern, and not everyone had the possibility of taking a class during their education to learn about software security. One company has software security as their specialty, and it was important that all employees knew the core principles of software security. To assist the developers building secure software, the company had created a set of routines that they had to follow.

A reassuring thought is that some Norwegian agencies possessing security critical information have a security department with formal knowledge and experience. They assist the developers that are often without security expertise in implementing the correct security measures. This is often necessary to handle the legal problems involved.

The lack of knowledge indicates that many security problems might be undetected, and that in some applications, the only form of "security" is that which is acquired through obscurity.

3.5. Untreated Concern

Almost all companies were worried about how their reputation would suffer if a vulnerability in their software became wide known, but when asked if it was enough to put an extra effort into securing their software, the answers were vague and non-committing. It is difficult to put a price on how much the company will suffer, but it is guaranteed to have a negative impact.

3.6. Customers' Take on Security

Companies dealing with external customers experience that about half of them are concerned about security. This is an indication that there are customers who are aware of some of the dangers out there.

The general impression of customers of IT services is that they are somewhat passive and uneducated in the threats that lurk in their domain. Banking and finance is forced to consider

software security when taking their services online. They have most certainly also learned from experience. Customers in other sectors, like the energy industry, are not too concerned with security, even though there are threats. Customers of software solutions will often opt out of security features if given a choice. One customer elegantly thwarted a security solution that one company forced into a product just because they did not like it.

3.7. The Weirdest Thing

Strangely, only a couple of the interview subjects had experienced concrete security breaches. The first critical idea was that the lack of audit trails, logging and intrusion detection systems (IDS) could have made it easy for a hacker to penetrate a system without being noticed. It is of course not something a company would want to advertise to the public, and it is quite possible that they take serious action to prevent such news from reaching the media. We suspect that security breaches are more prominent than given in the interviews, and that hackers easily can remove most of their traces.

3.8. Summary of Interview Findings

None of the companies had found or created any fully developed technique for integrating software security into agile software development. In our opinion it should have been an issue, but it would be naïve to think that companies would prioritize security over functionality without seeing it as a good return of investment. It is unknown whether upper management considers the potential loss if something is compromised.

The general opinion is that few developers are concerned with security, even fewer are formally trained. The organization is fully capable of sending their employees to courses, or encouraging learning software security in work hours.

The trend with securing with infrastructure alone seems to be recurring in most software solutions shipped. Many rely too heavily on the system administrators, assigning all responsibility for software security to them. The system administrators are often informed orally on how to protect the software. *Make sure it stays on the intranet* could be one such rule. If the system administrators all at once decided to leave, what would happen to these rules? The replacements might re-organize the infrastructure, and put a server on a public LAN, where it was never supposed to be.

As to how so many of the companies have managed to go without any remarkable break-ins or attacks, this is a mystery. As noted earlier, it might just be that they have not noticed them or that the media have not learned of them. Increased media focus on attacks that do happen may be what it takes for companies to be more concerned with software security. White-/gray-hat hackers might need to alert the media directly, as well as the company when they find vulnerabilities. Unfortunately, such news is probably not interesting for the mainstream media, and therefore the effect might be lost.

4. CASE STUDY: A DISTRIBUTED DEVELOPMENT EFFORT

Our case study (Sassoon, Jaatun, & Jensen, 2010) is based on the results of a European research project developing a healthcare platform. Since the platform deals with sensitive health data, it should comply with ("Directive 95/94/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data," 1995), which regulates the handling of private data for the member states of the EU. The Norwegian implementation of this directive was studied in order to define a set of security requirements to be included in the security design. As part of the evaluation, several components of the WS-* specifications were reviewed, in search of proper ways to deal with security for Web Services.

4.1. Project Characteristics

The case study project was funded by the European Commission's 6th Framework Programme, and thus had to satisfy a number of constraints. The project partners had to be diverse in geographical location and represent industry, academia and research institutions. EU research proposals all follow a set template, and it is implicitly assumed that the work will be organized in "Work Packages".

Although an agile Scrum approach was chosen for the project, the different work packages were established in a conventional manner, and developed independently. A separate security WP was a part of the plan, which should have been a good approach, since this would contribute to setting focus on the security aspects of the project. Unfortunately, due to a serious lack of continuity of key members in the project, the security design was delayed for a long period and it had to be elaborated in parallel with the implementation, and thus the implementation of the security mechanisms started before the security design was finished.

Other early project decisions contributed to the delay of the security design and the resulting parallelism mentioned above: No threat modeling was employed and no security requirements were thought of. The latter had to be done, finally, when the security design document was being developed.

The development process followed a Scrum approach for the most part. The security design process, however, ended up having more in common with a traditional waterfall approach, which may have contributed to the security work falling out of synch with the rest. In line with the chosen Scrum approach, a backlog of functional requirements was maintained. Somehow, only the functional results of the security design made it out of the backlog (e.g., the authentication and token management services were implemented), leaving most non-functional security aspects alone in the dark.

4.2. Assessment Results

The assessment proved that the proof-of-concept application and the middleware platform are vulnerable to common attacks targeting web applications. Considering the *OWASP Top 10 web application vulnerabilities* ("Top 10 2007 - OWASP,"), seven of them are present in our case study system:

1) Cross Site Scripting (XSS)
2) Injection Flaws
3) Information Leakage and Improper Error Handling
4) Broken Authentication and Session Management
5) Insecure Cryptographic Storage
6) Insecure Communications
7) Failure to Restrict URL Access

Based on our observations, we can infer that SOA-based systems in general are expected to suffer from the same problems if security is not treated properly. While this is not surprising, the fact that an organization that is concerned with data confidentiality and integrity does not implement basic security mechanisms makes us wonder how many other similar cases might exist.

Even though we evaluated a healthcare system, we can extrapolate the results to other domains since the vulnerabilities found are not specific. Therefore, the findings presented are relevant when considering the development of secure applications, based on SOA or not.

5. SECURITY EXTENSIONS TO AGILE METHODS

Both the literature and our empirical study show that there is a need for methods that ensure security issues to be taken care of during agile software development processes. In the following two extensions are suggested, both fitting well into an agile setting; Security backlog and Security-oriented TDD.

5.1. Security Backlog

When adding items to the backlog with the customer, the developers should spot security touchpoints and add them to a (possibly) separate security backlog, where each of the items has a reference to the product backlog items for which it is relevant. Here a separate backlog item is useful because a many-to-many relation can exist. Each item should also contain one or more misuse stories to describe how a person with malicious intent could do harm. Each item is prioritized according to risk[1]. The customer should be a part of the risk analysis, i.e., assigning (based on experience and intuition) H(igh), M(edium) or L(ow) to probability and consequence. Items with the highest risks are exposed to a more detailed and thorough process if the developers consider it necessary.

The intention of this method is to involve software security processes while not reducing the degree of agility (Keramati & Mirian-Hosseinabadi, 2008), thus its techniques should be lightweight. Exceptions are considered, and there should always be room for special cases on high-risk items. Consider using Microsoft Azure (Microsoft) when creating a new financial service on the web. At the time of writing, Azure is still considered new, and little is written on the security implications of having such services on a cloud platform. If the customer demands that the service should run on Azure, and none of the developers know enough about the security risks associated with cloud computing, then it should be considered an exception and extra effort should be put into making sure that the developers can secure it. This would probably involve having a spike[2] if using Scrum, and just learning as much as possible about the technology and security risks before proceeding. Of course, this reduces the degree of agility, but on high-risk items, security should outweigh functionality.

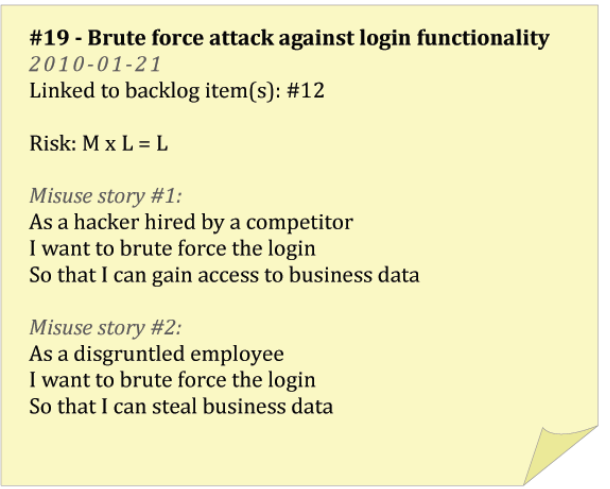
The following steps are based on how many Scrum projects are performed, and mainly contain extensions. Even though presented with focus on Scrum, the ideas are generic and agile enough to be used in other Agile methods.

5.1.1. Step 1: Requirements Gathering Phase

The customer and the developers are gathering requirements for a new system, and the customer provides various requirements, such as *The user must be able to log in*. This is a *security feature*. A developer points out a security threat he knows of, namely brute force attack. Stakeholders agree that this is a security issue, and the original requirement goes into the product backlog, while the newly identified security threats now go through a short detailing phase. Developers involve the customer in such a way that he/she understands the threat and is capable of evaluating at least the consequence of an exploit. The developers ask helpful questions like "*On a low/medium/high scale, how big is the consequence of a non-authenticated person gaining access to the system?"* and "*Are there competitors interested in the information within this system?"* The customer often knows the domain well, and is capable of giving an accurate consequence, and the developers know that the probability is linked to factors like competition, value of data inside the system etc. The customer and developers can now calculate the risk for the new security backlog item. Next step is to create at least one misuse story. Based on information from the customer, the developers are able to create an artifact like the one in Figure 1. Here it is important to note the artifacts are detailed as little as possible. They are to serve as reminders of what the concern was during the initial requirements phase. The technical details come in a later phase.

A special case worth mentioning is how to handle *secure code*. Another developer could have pointed out that the user input could be used to cause a *buffer overflow*, and that it is important to write secure code. It is on a higher level than a security feature, and it is something that often relates to all backlog items if the system is meant to be secure. We believe that this belongs in some sort of general policy that applies to the developers. The policy could specify rules as how to treat user input, and

Figure 1. Example of security backlog item artifact



should be updated when issues are spotted during the various detailing phases.

5.1.2. Step 2: Iteration 0

In iteration 0 the project starts up, teams are assembled and an initial architecture is laid out. The architect goes through the list of items on the product backlog to get an overview of the system. The architect goes through the list of items on the security backlog, and tries to form a picture of any architectural security features required. It is important that the chosen architecture does not impose restrictions on possible security features that might be needed later. For instance, the architect thinks it is a good idea to have a single access point in order to mitigate the brute force attack. This highlights the need for security patterns. All developers should evaluate the initial architecture together, as well as discussing how it holds up against the security requirements. There might be a need for discussing potential implementation issues. One developer might have spotted a problem with product backlog item n, which might interfere with security backlog item m. Discussing this in plenum ensures that every piece of combined knowledge is used, and many future problems can be avoided. An important feature of this method is the exception handling of high-risk items. If an item is marked as high risk, it should undergo an extra process where it is decided whether it should be thoroughly evaluated. This involves more rigorous security activities. Keep in mind that this might be needed to uncover everything about the item that is relevant in order to make sure it is properly secured when implemented. This requires that someone in the team has knowledge of a security process, or that someone is hired for this specific purpose.

5.1.3. Step 3: Sprint Planning (Each Iteration)

When the development cycle starts, developers pick the top prioritized backlog items and start detailing them in the beginning of each iteration. In a Scrum process, this is called a sprint-planning meeting. During this detailing-phase, the developers know there are items on the security backlog that is linked to the product backlog items they have picked. For example, one product backlog item may have a couple of security backlog items linked to it. This means that when creating more detailed user stories and acceptance criteria for the backlog items, they have to consider the security implications as well. A simple solution is to integrate the

security backlog items into the sprint backlog items. When integrating, create an acceptance criterion that can be used to verify that the threat is mitigated, and one or more security focused user stories. The process of consciously thinking of software security means that the developers are more aware of the security aspects of the tasks. The misuse stories from the security backlog items should also extend the sprint backlog item. The detail level on all artifacts increases in this step, and elicitation of misuse stories must be more precise and detailed (Peeters, 2005).

The goal is to have a simple process with as few extra artifacts as possible. If the developers just become more aware of security, then there is already a gain from the process.

5.1.4. Step 4: Implementation

During the implementation of the items on the sprint backlog, developers now have a detailed description of user stories, misuse stories and acceptance criteria to help them correctly implement the item. What it comes down to now, is the developer's knowledge and experience when it comes to software development and software security. Testing should be used to verify that the acceptance criteria are achieved during this phase. Best practice would be to implement each item using TDD and ensure that each user story and misuse story are implemented. This is good agile practice as it forces loosely coupled design, which in time requires less maintenance, is more robust and flexible. The tests also serve as documentation in respect to how things have been implemented. The idea is that writing tests first forces the developer to learn about the security threat he tries to protect against. Of course, the test-code can be poorly written, and not actually test that the system is protected. Here the collective code ownership rule might help. When having code-reviews, poorly written tests should be detected, and team members with more security expertise might know a more effective test to confirm that a given threat has been mitigated.

5.1.5. Step 5: Verification

When the system is nearing completion, penetration testing should be used to verify that it resists attacks as intended. A rigorous and well-performed penetration test of the system can expose parts that are not secure enough. If the testing reveals vulnerabilities, one sprint must be held to fix all these. All the tests written previously should of course pass, and should in theory be proof that the security backlog items have been considered and implemented. It should be confirmed that each item on the security backlog has been included in the process. Use of static code analysis tools is encouraged, since it can uncover common programming mistakes and potential problems.

5.2. Security-Oriented TDD: Security Tests

Robert C. Martin (Martin, 2008) provides a short summary of the workflow in the three laws of TDD:

- *You are not allowed to write any production code unless it is to make a failing unit test pass.*
- *You are not allowed to write any more of a unit test than is sufficient to fail [...].*
- *You are not allowed to write any more production code than is sufficient to pass the one failing unit test.*

This workflow works very well for implementing functional requirements. Our goal is to adjust this workflow to suit testing of security features, and attempt to keep it applicable to most agile methodologies.

We assume that the planning phase has resulted in functional requirements and security requirements. In addition, all requirements are already detailed, and the security requirements (in form of misuse stories[3]) are placed in the security backlog (see section 5.1). All these items are required in order to write good tests. Here it is again important to point out the

difference between "security features" and "secure features"; that the former should be able to resist an attack might seem obvious, but it will require significantly more imagination and inspiration to describe misuse stories for "ordinary" features.

What follows is an example of a workflow: When a developer picks a task to work on, he should at the same time retrieve and review the related security backlog items. Not all tasks need have security requirements, but in our example they do. Usually, a developer would start working directly on the functional requirements, and writing tests before implementing production code. Depending on the type of feature that is to be secured, the developer must decide whether to write a unit test or a security test first. This is something the developer can decide by intuition with some training. The best thing would be to always write the security tests first, and then shape both implementation code and unit tests to support the security tests. Unfortunately, this is not always feasible. If developers were to follow the standard TDD, they would need to "break the rules" in order to make the test pass. It is not always possible to write a security test before there exists production code to secure.

A security test aimed at verifying that only authenticated users have access to the admin page will always pass if nothing has been implemented. If the test tries to load a non-existing page, the result could be an HTTP code 404 (Not Found), and the user by definition does not have access to the admin page. To mitigate this, the rules must be bent a bit. The sequence in which the developer writes unit and security tests must be adapted to the situation. It is important that the developer try to get back on *TDD-track* as soon as possible, and follow the recommended workflow. The problems only occur as the first tests for new functionality are written.

Developers should strive to verify that important security requirements are implemented and that the functional requirements are protected sufficiently. To ensure this, each functional requirement has one or more security tests. A security test attempts to verify that a specific security requirement is implemented and protects against an identified threat. This includes one or more test attempts to exploit parts of the systems for a vulnerability described in the security requirement. Where applicable, test-permutations (e.g., through use of fuzzing) should be used to uncover weird boundary conditions, possible overflows etc.

6. DISCUSSION

The core idea of having a security backlog is that security should become more of a concern for the developers than it is today. This is a lightweight method with very little overhead. The intention is that it will not scare off developers as some of the other humongous security processes might. There are few new things to learn, and the customer can easily be included. This method depends on having at least one person in the development team with software security competence, or having a budget that allows hiring in someone for the job. The efficiency of this method depends on how quickly developers uneducated in software security can pick up the new mindset, learn specific techniques to avoid security holes, learn where to look up known problems and learn how to learn from others. If developers are unwilling to change from their regular routine where programming is just a straightforward task of "making things work", then this approach will fail.

Security-oriented TDD, like security backlogs, is a way of getting developers more engaged in software security. Being aware that there are security touchpoints, and having defined threats to protect against, might very well give good results in practice. Developers can verify that misuse stories are countermeasured. These tests are readable for low-tech stakeholders, and they can verify that they agree on how the system is tested. Newcomers can quickly read the tests and see what is tested (and how it is tested).

There might not be a need for defining security specific tests, as (Boström et al., 2006)

have shown that abuser stories can be translated into security-oriented user stories, which in theory should be testable. However, problems occur when following a strict TDD workflow. When there is no production code, writing tests to protect it might be useless. Therefore, the developer must choose his own way of solving the problem.

When it comes to the case study, we have to take into account that both the proof-of-concept application and the middleware platform were prototypes of an ongoing research project, and are still not ready for production. Even so, this is not an excuse for the apparent relaxed focus on security aspects, considering that the formal plans maintained a high security posture. Nevertheless, the costs for fixing the issues at this point in the project are certainly higher than if the assessment was performed earlier, or if security testing had been part of the secure development lifecycle (SDLC).

We can wonder about the project aspects that may have influenced the security achieved and perceived. Is Scrum the problem? Is it Waterfall? Or is it simply a communication problem? As the original project plan did not comprise testing, no problems could be discovered and associated to a particular moment in time.

Is the idea to implement a separate security work package a good one? Work packages are typically enough unto themselves, evolving on their own while ignoring other WPs. Is it better to include security in every work package? We have to consider that there are re-usable security "components", and these are probably best developed in a separate work package. Furthermore, a separate security WP gives security the proper attention, avoiding a project falling into the usual trap: "*We'll take care of security AFTER everything else works"*.

McGraw argues that security needs to be in focus from the beginning (McGraw, 2006), and that the focus should continue during the whole project. The fact that the security design was delayed and, therefore, other components were developed without considering the security work package, set the stage for a big hole in the platform. Communication problems among project members intensified the issues, by not bringing word about the integration of the results from the security WP and the consequences related to their (non-) use. According to (Lipner & Howard, 2005), there is a need for a *security push* in the whole organization, or project groups, in order to focus on security and identify problems.

Security requirements were not part of the project requirements. Partly using a Model-Driven Development (semi-agile) approach, the system design was based on models/diagrams, such as use cases, from which functional requirements were derived. The use cases in question did not cover security, and thus no security requirements were generated (we would have expected some obvious ones, such as confidentiality-protection of a doctor-patient message). Employing misuse cases would have been a good idea in this setting, but they were voted down early in the project.

Although agile methods make it difficult to comply with the stringent documentation requirements of, e.g., the Common Criteria (*Evaluation criteria for IT security Part 1: Introduction and general model* 2005)[4], several authors have argued that agility and security need not be inversely proportional measures. (Beznosov, 2003) opines that the agile XP methodology can provide "good enough" security, while (Wäyrynen, Boden, & Boström, 2004) claim that the solution to achieving security in an XP development is simply to add a security engineer to the team. (Siponen et al., 2005) advocate a solution that more or less can be summed up as "think about security in every phase".

(Poppendieck & Morsicato, 2002) argue that agile methods (specifically: XP) are just as suitable as traditional development methods for developing safety-critical applications. It may not follow immediately that "safe" software is also "secure", but the former is required to pass auditing procedures that should be customizable to suit requirements for the latter.

Throughout the conducted interviews, we noticed that many of the professionals within the field of software security use several terms

in an ambiguous fashion. For instance, when some papers talk about Microsoft's STRIDE method, they refer to it as a risk analysis tool, even though Microsoft themselves refer to it as a threat modeling tool. There is also a lack of clear definitions, even for something as fundamental as "security requirements". For newcomers, this field gives an impression of being immature and disorganized. This problem has been recognized by academics and professional for quite some time, but they do not know how to fix it. We believe that the secure software engineering community must decide on a clearly defined terminology as a first step toward maturity.

Although it might be tempting to suggest that a project where security is vital should be performed using a formal top-down project methodology, we have to face the fact that software development is becoming more and more agile, and there should be a way of mitigating the security drawbacks of using agile methodologies. This might be projects where security is vital (e.g., military, finance, health), or projects where security is not a concern at first, but emerges as the software evolves.

An excellent example is Adobe Acrobat – a PDF-document editor/reader. The initial version was intended for creation and reading of PDF files, and not much else. Security was probably not a big concern, because who could exploit documents following the PDF-standard? However, in version 3.02 of Adobe Acrobat, JavaScript features were added to the application. This new functionality was soon exploited (Narraine, 2006), and was the first of many security problems with Adobe Acrobat. They have now added support for viewing 3D-objects, playing Flash files, viewing CAD-files and third-party plug-in support. CERT[5] reports 25 security vulnerabilities related to Adobe Acrobat per Feb 17th, 2010. This example supports the notion that all software should boast secure features, i.e., even code without specific security features should be un-exploitable.

Unfortunately, there is no silver bullet for making software secure - it is all about knowledge. There is no obvious way of ensuring that security is taken care of in agile software development, but an important fact many tend to neglect is that most agile methods require experienced developers for optimal performance. With sufficient experience combined with the concept of collective knowledge in agile methods, project participants might spot security issues as they occur if one or more of them have training in software security. Another important issue with agile software development is that few of the methods have practices for rigid testing of security. In the waterfall model, there is a dedicated phase named verification, which is used to verify that the software behaves as the customer wants. This includes various types of both automated and manual testing.

7. LIMITATIONS

Our empirical study was based on agile teams working closely together in one location, while the case study was a project performed by a distributed agile team. This difference makes it difficult to draw conclusions covering both cases.

It is also difficult to generalize from the case study as EU projects do not produce production-quality code. However, there were explicit security objectives in this project, and that is why it is still an interesting project for us to analyze.

The case study has been approached from a single viewpoint, and should ideally have been augmented with in-depth interviews of more project participants.

The suggested method extensions have not been tested and evaluated, which will be a natural part of further work in this area.

8. FURTHER WORK

There is a lack of empirical knowledge regarding the relative security benefits of agile development vs. conventional (e.g., waterfall) development practices. We would like to conduct a larger study, comparing the degree of software security resulting from different

development methodologies. This would entail tackling several non-trivial problems, such as defining how to measure "software security" in a given piece of software, and how to compare software products that by necessity must be quite different.

In this project we have been studying organizations using the Scrum methodology. It would be interesting to study organizations using the agile components of the Microsoft Secure Development Lifecycle (Sullivan, 2008) to see whether this methodology changes developers' mindset and increases the focus on software security throughout the complete development cycle.

9. CONCLUSION

The interviews and case study presented in this article suggest that it is necessary that every person involved in a project is aware of the consequences of not thinking about, implementing and testing security from the beginning. Only then will it be possible to achieve more secure systems.

Combining software security with agile software development appears to be difficult to do in an elegant way without any compromises. Our suggested solution is to integrate parts of security activities into any suitable agile activities, while trying to figure out the pain threshold for when the reduction of agility becomes too large.

This article has suggested two possible extensions to the agile Scrum method. These extensions are an attempt to take the edge off some of the incongruence between secure software development and an agile mindset. Both Security backlog and Security-oriented TDD are lightweight methods that do not require much documentation and artifact production.

ACKNOWLEDGMENT

Thanks to Jostein Jensen for his valuable input on earlier phases of this research.

REFERENCES

Beznosov, K. (2003). *eXtreme Security Engineering: On Employing XP Practices to Achieve "Good Enough Security" without Defining It.* Paper presented at the First ACM Workshop on Business Driven Security Engineering (BizSec).

Beznosov, K., & Kruchten, P. (2004). *Towards Agile Security Assurance.* Paper presented at the New Security Paradigms Workshop, Nova Scotia, Canada.

Boehm, B., & Basili, V. R. (2001). Top 10 list [software development]. *Computer*, *34*(1), 135–137. doi:10.1109/2.962984

Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K., & Kruchten, P. (2006). *Extending XP practices to support security requirements engineering*. Paper presented at the Proceedings of the 2006 international workshop on Software engineering for secure systems (SESS '06).

Cheswick, B. (1990). *The Design of a Secure Internet Gateway*. Paper presented at the USENIX Conference.

Directive 95/94/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. (1995). Retrieved October 6, 2008, from http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:HTML

EPTA. (2006). *ICT and Privacy in Europe, Experiences from technology assessment of ICT and Privacy in seven different European countries*. Retrieved September 23, 2008, from http://epub.oeaw.ac.at/ita/ita-projektberichte/e2-2a44.pdf

Ilioudis, C., & Pangalos, G. (2001). A framework for an institutional high level security policy for the processing of medical data and their transmission through the Internet. *Journal of Medical Internet Research*, 3.

ISO. (2005). *Evaluation criteria for IT security Part 1: Introduction and general model* (Tech. Rep. No. 15408-1). Geneva, Switzerland: ISO/IEC.

Keramati, H., & Mirian-Hosseinabadi, S. H. (2008). *Integrating software development security activities with agile methodologies*.

Lipner, S., & Howard, M. (2005). *The Trustworthy Computing Security Development Lifecycle*. Retrieved from http://msdn2.microsoft.com/en-us/library/ms995349.aspx

Martin, R. C. (Ed.). (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Prentice Hall.

Mavridis, I., Georgiadis, C., Pangalos, G., & Khair, M. (2001). Access Control based on Attribute Certificates for Medical Intranet Applications. [JMIR]. *Journal of Medical Internet Research*, *3*(1). doi:10.2196/jmir.3.1.e9

McGraw, G. (2006). *Software Security: Building Security*. Reading, MA: Addison-Wesley.

Microsoft. (n.d.). *Windows Azure Platform*. Retrieved February 19, 2010, from http://www.microsoft.com/windowsazure/

Narraine, R. (2006). *Hacker Discovers Adobe PDF Back Doors*. Retrieved from http://www.eweek.com/c/a/Security/Hacker-Discovers-Adobe-PDF-Back-Doors/

OWASP. (2007). *Top 10 2007*. Retrieved July 10, 2008, from http://www.owasp.org/index.php/Top_10_2007

Peeters, J. (2005). *Agile Security Requirements Engineering*. Paper presented at the Symposium on Requirements Engineering for Information Security.

Poppendieck, M., & Morsicato, R. (2002, September). XP in a Safety-Critical Environment. *Cutter IT Journal*, *15*, 12–16.

Sassoon, R., Jaatun, M. G., & Jensen, J. (2010). *The road to Hell is covered with good intentions: A story of (in)secure software engineering*. Paper presented at the 4th International Workshop of Secure Software Engineering (SecSE 2010).

Scrum Alliance, I. (2009). *What is Scrum*? Retrieved March 23, 2010, from http://www.scrumalliance.org/learn_about_scrum

Siponen, M., Baskerville, R., & Kuivalainen, T. (2005). *Integrating Security into Agile Development Methods*. Paper presented at the Hawaii International Conference on System Sciences, HI.

Sullivan, B. (2008). Agile SDL: Streamline Security Practices for Agile Development. *msdn Magazine*. Retrieved from http://msdn.microsoft.com/en-us/magazine/dd153756.aspx van der Haak, M., Wolff, A. C., Brandner, R., Drings, P., Wannenmacher, M., & Wetter, T. (2003). Data security and protection in cross-institutional electronic patient records. *International Journal of Medical Informatics, 70*(2/3), 117-130.

Wäyrynen, J., Boden, M., & Boström, G. (2004). Security engineering and eXtreme programming: An impossible marriage? In *Proceedings of the Extreme Programming and Agile Methods - Xp/ Agile Universe 2004* (Vol. 3134, pp. 117-128). Berlin: Springer Verlag.

ENDNOTES

1. Risk = Probability x Consequence
2. A time-period set aside to experiment and learn something unknown in a user story.
3. Similar to user stories, but consider them textual versions of misuse cases.
4. Assurance level 4 and less can be verified for legacy systems not developed with CC evaluation in mind.
5. www.kb.cert.org

Torstein Nicolaysen is an MSc student in Computer Science at the Norwegian University of Science and Technology (NTNU) . After the completion of his MSc thesis, he will start working for BEKK Consulting. His research interests include making software security more agile, software security in general, Agile methodologies, software development and craftsmanship.

Richard Sassoon recently completed his MSc in Information Security as part of the Erasmus Mundus NordSecMob programme at NTNU and the University of Tartu. His interests include software engineering, software testing (including security testing), and software security with focus on web security. He is currently employed as a software developer at MVisjon AS.

Maria B. Bartnes Line received her MSc degree from the Norwegian University of Science and Technology in 2002, and has since been employed as a research scientist at SINTEF ICT in Trondheim. Her research interests include software security, privacy, intrusion detection and incident response. She is the manager of the information security research group.

Martin Gilje Jaatun graduated from the Norwegian Institute of Technology (NTH) in 1992, and has been employed as a research scientist at SINTEF ICT in Trondheim since 2004. His research interests include software security "for the rest of us", information security in process control environments, and security in Cloud Computing.