LAUREA
UNIVERSITY OF APPLIED SCIENCES
Prime Mover

Security Model for Agile Software Development

Koistinen, Pasi

2013 Laurea Leppävaara

Laurea University of Applied Sciences
Laurea Leppävaara

Security Model for Agile Software Development

Pasi Koistinen
Security Management Degree
Master's Thesis
October 2013

Laurea University of Applied Sciences
Leppävaara
Degree Programme on Security Management

Abstract

Pasi Koistinen

Security Model for Agile Software Development

Year 2013 Pages 111

Nowadays information systems, applications and architectures are subject to a rapidly changing environment. Business requirements and cyber threats are the de facto environment of software development these days. Agile software development methods have become popular because they promise to be adaptable to changes in the environment.

Traditional deterministic security management models are applied in waterfall software development approach. In traditional software development approaches, security requirements are planned ahead of development work, and implemented according to the plan. The problem is that static security models are unable to respond to rapid changes in the requirements and threats during the development. Furthermore, applying the traditional static security models to Agile Methods has proven to be difficult since these methods were not initially designed to be compatible with iterative development process models. Real-life outcomes of software development security have also proven to be a tragic story since majority of web applications are highly vulnerable to cyber attacks. Software security should be integrated into the development process, but this is usually not the case.

This thesis presents an Agile Software Model based on the Agile Manifesto, the core theory of Agile software development. The model integrates universally applicable security mechanisms into the Agile development process. The most important security mechanisms in this model are designed to be as an integral part of the development process instead of being external to the development.

Organizations in Agile software development are able to use the Agile Security Model for developing secure software for their customers. The model also provides security assurance for meeting the compliance requirements of the customers.

Keywords: Agile software development, information security, security model, Agile software security

Laurea-ammattikorkeakoulu **Tiivistelmä**
Leppävaara
Turvallisuusosaamisen koulutusohjelma

Pasi Koistinen

Agiilin sovelluskehityksen tietoturvallisuuden hallintamalli

Vuosi 2013 Sivumäärä 111

Tietojärjestelmät, sovellukset ja niiden arkkitehtuurit elävät nykyään muuttuvissa ympäristöissä, jossa liiketoimintavaatimukset ja uhkakuvat muuttuvat nopeasti. Agiilin, eli ketterän sovelluskehityksen menetelmät ovat tulleet suosituiksi, koska niiden avulla kyetään vastaamaan muuttuviin vaatimuksiin perinteisiä staattisia menetelmiä paremmin.

Sovelluskehityksen perinteinen tietoturvallisuuden hallinta ja menetelmät ovat perustuneet deterministisiin malleihin, kuten vesiputousmalliin, jossa tietoturvavaatimukset määritellään ennalta ja toteutetaan suunnitelman mukaisesti. Staattiset tietoturvamallit eivät kykene vastaamaan nopeasti muuttuviin vaatimuksiin ja uhkakuviin nykyajan muuttuvassa ympäristössä mielekkäästi. Staattisten tietoturvamallien soveltaminen notkeaan sovelluskehitykseen on myös osoittautunut ongelmalliseksi, koska perinteisiä tietoturvamenetelmiä ei ole kehitetty käytettäväksi sovelluskehitysprosessissa, jossa kehitys etenee sykleissä kuten agiilissa sovelluskehityksessä. Sovelluskehityksen tietoturvamallien tehokkuus on osoittautunut kyseenalaiseksi, koska useimmat verkkosovellukset ovat myös osoittautuneet erittäin haavoittuviksi. Sovelluksia ei ole lähtökohtaisesti suunniteltu ja tehty tietoturvallisiksi.

Tässä työssä esitellään notkean sovelluskehityksen tietoturvamalli, joka perustuu notkean sovelluskehityksen perusteoriaan, Agile Manifesto:on. Tietoturvamallissa käytetään menetelmiä, jotka soveltuvat agiiliin sovelluskehitykseen ja integroituvat osaksi sovelluskehitysprosessia sen sijaan että tietoturvallisuus olisi irrallinen tai ulkoinen osa kehitystyötä. Tietoturvamallin avulla agiilia sovelluskehitysmenetelmää käyttävät organisaatiot voivat tuottaa asiakkailleen turvallisempia sovelluksia sekä osoittaa, että asiakkaiden tietoturvavaatimukset on huomioitu ja toteutettu kehityksessä toivotulla tavalla.

Asiasanat: Agiili, notkea sovelluskehitys, tietoturvallisuus, hallintamalli, tietoturvallinen sovelluskehitys

Contents

1 Introduction 7
1.1 Reflections on Software Security 8
1.2 Scope and Limitations 9
1.3 Structure of the Report 10
2 Silverskin Information Security LLC 10
3 Core Concepts 11
4 Security Problems of Agile and Justification of Work 12
4.1 Failed Software Development Projects 13
4.2 Lack of Trust in the Software Development and its Outcomes .. 14
4.2.1 Incompatibility of Traditional Assurance Methods 16
4.2.2 A Dangerous Outcome: Vulnerable Software 18
4.3 The Benefits of an Agile Security Model 25
5 Research Process 25
5.1 Constructive Research Approach 27
5.1.1 Philosophy and Theory of Constructivism 27
5.1.2 Constituents of the Constructed Agile Security Model 29
5.2 Phases of Construction 31
5.3 Research Methods and Data Collection 33
5.3.1 Review and Analysis of Literature 33
5.4 Critique of Normative Approaches in Software Security Development in Literature 35
5.5 Interviews of Leading Agile Security Experts 38
6 Traditional vs. Agile Software Development Approaches 41
6.1 Agile Software Development 41
6.2 Comparison of Control Mechanisms in Agile and Traditional Approaches 43
6.3 Sequence of Software Development Phases 48
6.4 Short Explanation of Scrum Method 51
7 Analysis of Security Problems and Applicable Security Mechanisms 52
7.1 Identification of Applicable Security Mechanisms 55

7.1.1 Results of Literature Review 56
7.2 Analysis of Applicable Security Mechanisms 59
7.3 Detailed Description of Applicable Security Mechanisms 66
7.3.1 Initial Planning Phase 66
7.3.2 Planning & Requirements Phase 71
7.3.3 Architecture & Design Phase 80
7.3.4 Development & Implementation Phase 83
7.3.5 Testing & Evaluation Phase of the Project 86
7.3.6 Delivery Phase 90
8 The Agile Security Model 92
9 Conclusions 94
9.1 Reflection About the Success of the Research 95
Tables 104

1 Introduction

Many organizations develop software using Agile Methods, including very massive ones like Microsoft, Cisco, Raytheon, General Motors, Merril & Lynch and numerous others (CIO Magazine, 2004). Historically speaking software security has not been adequately included in the Agile software security development methods. Agile Methods place emphasis on customer communications and rapid feature development instead of meticulous preliminary planning. On the other hand, security is a priority for customers and should not be overlooked because of regulatory and privacy requirements of today's highly interconnected world.

There is a concern that Agile Methods do not create secure code (Beznosov & Kruchten, 2004-2006). There is also a concern that Agile development Methods lack reliable methodologies and explanations for creating secure software (WASC, 2008 & Silverskin, 2012). This implies that there is a gap between the customer's security requirements and the secure end results: The customer is unable to perceive the causal relations that would explain whether security is adequately addressed or not (Peeters, 2005).

To aggravate matters, there is little Agile security expertise available on the market today (Interviews of Agile Experts A & C). The first step of remediation is the acknowledgement of current state of affairs and then integrating security into Agile software development Methods. The goal of this research is to answer that need.

Other Agile secure software development models do exist, some of them utilizing a process-oriented approach (Microsoft, 2009), some with a focused scope (Peeters, 2005), and some with a control-based approach (OpenSAMM, 2013 & BITS, 2012). Howevern, the most common obstacle against integrating security into the Agile development process stems from security being external, exogenous, to the development process (Sillitti & Succi, 2008). This leads to the problem that security activities may be easilly skipped, or they make

no sense to the developers. Therefore it was necessary to develop a model that would be based on Agile Methods, utilize effective security mechanisms that integrate into the development process, and at the same time not encumber the developers with tasks that make no sense to them.

1.1 Reflections on Software Security

Speaking generally we need to comprehend the concept of emergence before we can comprehend how the emergence plays a role in software security. Therefore, an example is in order: in physics, pressure of gas is proportional to its temperature. Pressure is an emergent phenomenon that can be explained more precisely on microscopic (atomic or molecular) level as the cumulative mass effect of movement and momentum of many tiny gas particles. In fact, on the microscopic level, the emergent quality of pressure does not physically exist! Quite similarly, we can measure emergent feature of security appear from a seemingly non-deterministic, chaotic Agile software development process: it seems chaotic because there is a lack of explanation for the underlying rules of action between the atomic particles. In other words, we need to understand which security mechanisms are effective and applicable to Agile before we can explain how the macroscopic phenomenon of emergent software security arises. Also in security, as in any subject of real world, some energy and effort are needed to change the state of the system into something else, something secure. This first rule of thermodynamics applies to pressure and security likewise: unless energy, or time and effort, are invested nothing is bound to change.

Theoretically software security can be perceived as an emergent qualitative feature of software development processes. Such emergent quality arises as a result of diligent development work in which each atomic feature of the software is carefully conceived, designed, written and perfected in practice keeping in mind the importance of each security aspect to the customer who bears the cost. Conversely the emergence implies that a priori security of a large

system cannot be precisely defined and proven, and that the level of software security is always relative to the energy or investment appointed to the security portion of the development. The opposite feature of security is insecurity, which is almost guaranteed to emerge if the aforementioned diligent development approach is neglected. Far too many applications seem to contain serious security weaknesses (Silverskin, 2012 & WASC, 2008).

The metaphor of emergence fits well with Agile software development because Agile is often perceived as non-deterministic process with hard-to-predict end results.

1.2 Scope and Limitations

An assessment of Agile security problems was conducted in order to identify the inherent problems in Agile software development.

Since this research is focused on Agile development security we need to make clear distinction which development methodologies and approaches are within this scope and which ones are out of it. For example, traditional waterfall development methods should evidently be out of the scope of this work, unless we need to briefly address them to show their differences in contrast with Agile approaches. With regards to various different flavors of Agile (different Agile variations), we do not want to concentrate on any specific one while forgetting all the rest. All Agile approaches have a lot in common, and therefore we need to propose solutions that are general enough to work sufficiently in most, if not all, Agile variants.

Finally, any detailed confidential information related to Silverskin, its customers, interviewees or third parties is out of the scope of this report.

1.3 Structure of the Report

This research consists of the following main chapters:

- Description of the organization where this research was conducted and where the Secure Agile Model was developed.
- Core concepts and the research process. We use Agile manifesto as a framework to construct an Agile Security Model.
- Traditional software development methods and control mechanisms are compared against Agile development approaches. Here we learn to understand the strengths and weaknesses of both approaches.
- Security problems in Agile and justification of the work–chapter explain several reasons why this research is necessary and beneficial. Software projects fail all too often, customers do not trust that Agile Methods could create secure software, traditional assurance methods are not compatible with Agile, and software is infested with vulnerabilities.
- In the next chapter, an analysis of security problems is performed against the principles of Agile manifesto. Each identified problem is broken down into smaller sub-problems. Next, applicable security mechanisms are proposed for each sub-problem based on interviews and analysis of literature.
- In the final chapters we construct and present the Agile Security Model. The model integrates the applicable security mechanisms into a generic iterative Agile software development framework.

2 Silverskin Information Security LLC

This research has been conducted in the company where I am currently employed. The company, Silverskin Information Security LLC, focuses on solving demanding information security assignments for its customers. The customers are other businesses and organizations (not consumers).

The company has been in the business of security engineering, web application security and security management consultancy since 2009. The idea to write this thesis began from our observations from web application security auditing. We observed problems in our customer's software development projects because we had a view into the whole process of software security process from the beginning of the projects until the final end results: often we started working closely with the customers in order to conceptualize, design and build a secure system, and in majority of cases we also performed security testing and auditing of the resulting software. Our experience from the application security auditing has given a reason to understand the root causes behind the vulnerabilities in software. Furthermore, after having done hundreds of such assignments our insight has covered the area of software security from a very wide perspective.

This research was deemed necessary in Silverskin because the company wanted to develop their services to customers in the Agile development business. Many customers seemed to either struggle with creating secure code, or their software was infested with security vulnerabilities. This led us to inquire whether lack of security mechanisms in the customer's software development process could be the reason for their problems. We decided to research and develop a secure software development model to improve things. This work implied that we tested various security mechanisms in customer projects and to verify if those mechanisms proved to be useful in practice. Because the scope of the research covered a wide range of software development process phases, the research took two years to finish.

3 Core Concepts

The following core concepts are used throughout this research. Additional related concepts are included in attachments. For example, Firesmith (2003) has also created an information model for security engineering that explains relations of several related concepts (figure 20 at the end of this report).

Agile: Agile Methods (AM) are a set of software development methods and techniques. They have been conceived to deliver their results timely, on budget, and with high customer satisfaction. According to De Lucia & al. (2008, p. 252) “AMs emphasize the human factor in the development process and the importance of direct, face-to-face communications among stakeholders, the value of simplicity, perceived as an elimination of waste, and continuous process improvement, as the transposition to the Software Industry of Total Quality Management.”

Attacker story (synonym for abuser story or misuse case): A description of an unwanted event that could threaten the confidentiality, integrity or availability of customer’s data stored, processed or transmitted by the application. The story can be initially described as a high level requirement, and is later broken down into technical development tasks.

Backlog (sprint or product backlog): “The sprint / product backlog (or "backlog") is the requirements for a system, expressed as a prioritized list of product backlog items. These included both functional and non-functional customer requirements, as well as technical team-generated requirements” (Scrum Alliance)

Software security requirements: a specification of security mechanisms (controls) that eliminate or reduce the likelihood that the software contains vulnerabilities.

Security mechanism: mechanism that eliminates or mitigates (reduces) the impact that exploitation of vulnerabilities may cause. It is a synonym for software security control. (Firesmith, 2003 p. 33)

4 Security Problems of Agile and Justification of Work

This chapter explains several identified problems in Agile software development methods and explains why it was deemed necessary to conduct this research.

4.1 Failed Software Development Projects

According to Beznosov & Kruchten (2004), for decades developers of software have wanted to use the technical-rational approach to software project management. This approach implied sequential project lifecycle (waterfall model) with beforehand planning and design. Constant monitoring was used to force the project to conform to the plan. They note that this approach has been successful in certain aspects: for example having designs and plans available early in the project has enabled the certification of information systems against external assurance standards: e.g. safety certification in avionics and medical instrumentation. This has also made acquisitions of information systems easier for the buyer.

Beznosov & Kruchten (2004) and De Lucia & al. (2008) all note that the technical-rational approach is flawed because the success rate of software projects has proven to be low, with less than 50% success (Standish Group, 1994). Most of the failures are due to management practices in software projects (table 1). The main reason why Agile Methods were originally invented was that many of the management paradigms adopted from construction and manufacturing simply do not work in software development domain.

Study by Standish Group (1994) focused on the reasons for failed software development projects (table 1). For example the second largest reason, according to the study, is low customer involvement with 12.4% of failed projects. Choosing an Agile development methodology could mitigate this risk since key principle of Agile is to maintain high level of customer involvement. Agile approach could also prove useful which applies to lack of management support with 9.3% share since Agile principles should also address this project risk. Furthermore risk of changes in (software) requirements (8,7%) should also be

mitigated by choosing an Agile approach over traditional ones. In fact, a very large portion of software project failures could have been avoided by developing the software with Agile approaches. This is good reason to focus this research on secure Agile software development (instead of a generic approach, including traditional development methods).

TABLE 11.2. Main Causes of Project Failure

Problem	%
Incomplete requirements	13.1
Low customer involvement	12.4
Lack of resources	10.6
Unrealistic expectations	9.9
Lack of management support	9.3
Changes in the requirements	8.7
Lack of planning	8.1
Useless requirements	7.5

Table 1: Main causes of software development project failure (Standish Group, 1994)

4.2 Lack of Trust in the Software Development and its Outcomes

Not only do software projects often fail but they also produce insecure software products. During a period of two years, Silverskin performed security audits against a sample of 130 web applications. A research was conducted to make a joint effort regarding the results of the audits. The main finding is that the tested applications appeared to be infested with severe vulnerabilities. A significant portion of the inspected applications and data within the systems were poorly protected and could have been compromised because of the poor level of application and system security (Silverskin, 2012). Luckily in a few instances the research data also contained cases where the tested applications seemed to be extremely secure.

Many of the customers of Silverskin have adopted some sort of an Agile method for developing software. The inherent problem is that the Agile software development models currently do not holistically and causally explain how and why security should be addressed in the development process. Hence the customers who pay for the development work are often unable to understand how Agile process could produce secure end results that meet the compliance requirements of the customer. Agile Security Expert A stated in interview that: ”there are very few experienced people available on the market who understand secure Agile development”.

The customers may also be unable to verify if the end results of the Agile development are secure enough for their business needs and requirements. According to Beznosov & Kruchten (2004) the Agile software development approach appears totally contrary to traditional assurance practices ”…in system certification, independent validation and verification, and in software acquisition practices…”. Consequently the customers may be unable to verify if their security compliance requirements have been achieved or not.

A new study conducted by Ponemon Institute and Security innovation (2013) also paints a bleak picture of adoption of secure software development methodologies. The main findings of the study were that:

- Organizations do not have a defined software development process
- Organizations are not testing for security
- Policies and requirements are ad-hoc and not integrated into the SDLC
- Organizations do not have a (security) training program in place
- Organizations do not measure compliance with regulations and standards
- Organizations do not understand application security risks
- There is a disconnection between the executives and practitioners regarding the maturity of the application security and activities

According to Baskerville (2004), there is also a gap between the information systems and security that creates a flood of vulnerabilities unless new approaches are developed for rapid and continuous development of security safeguards.

In conclusion:

1. Traditional software security assurance methods are mostly incompatible and do not work in Agile software development. The information gathered from Agile Expert interviews, literature and scientific research provides evidence that support this claim.
2. Applications that are the outcome of software development seem to endure faulty security design and are generally infested with software vulnerabilities. (This observation pertains to applications developed with both traditional and Agile Methods.)

4.2.1 Incompatibility of Traditional Assurance Methods

SAFECode (Software Assurance Forum for Excellence in Code) describes Agile as a development technique that inherently contains a kind of an ”early feedback and error correction system”. They claim that Agile can potentially identify and repair software defects earlier in the development process than traditional development methods like waterfall. The security mechanisms that are used in traditional models are recognized to coalesce poorly with Agile since Agile development cycles (Sprints) are short, typically 2-4 weeks. Therefore according to SAFECode, software developers find it difficult to comply with a tedious list of various assurance tasks. It is deemed too burdensome to perform a long list of security tasks every few weeks. This is considered to be counter-productive. As a result, the developers often skip security tasks altogether and consequently develop software that contains vulnerabilities. The problem of security sometimes being a hindrance to software development is also acknowledged by Siponen, Baskerville and Kuivalainen (2005): Agile security Methods should not hinder the development project.

For example, Agile Expert A commented that security tasks should not be driven in the Agile process by requirements that are added on top of the development process (i.e. external, exogenous control). He made an example by criticizing a practice where security tasks are repeated at the end of every sprint. He made an example of such an Agile practice called ”Definition of Done, or DoD”. In his words, ”…if DoD is used to drive security work, there is a huge risk that the work will not be done because it is solely up to the development team whether such work will be done, or if they decide to skip it. If the development team says that the security activities on DoD have been executed, it could be true, or maybe it is not true. It is common in software business that there is a lot of business pressure to do things as fast as possible. Which are the first things that you remove under pressure? It is the items in the DOD list where developers remove their tasks because no one can tell how much time finishing those DoD tasks will take.” I believe that this observation is an indicative of mismatching security assurance methods between Agile and traditional approaches. This also describes the inherent weakness that exogenous control contains: the control provides low level of assurance because it is easily circumvented by its subjects.

Adding external controls (exogenous) on top of Agile development process creates practical problems. How should we approach this problem? Beznosov (2003) proposes that a number of adjustments and changes are necessary in software development security methods. The necessary changes are all related to the incremental nature of Agile. For example, risk analysis, vulnerability analysis, security testing and use of test automation suites should be redesigned in order to work in incremental development processes.

Beznosov & Kruchten (2004) have also analyzed the suitability of security assurance methods of traditional software development approaches against Agile Methods (table 8). They make a cross-reference of assurance methods that are either compatible, work independently of development approach, can be semi-automated, or if there is a ”mismatch” (incompatibility) between tradi-

tional and Agile assurance methods. The result of the cross-reference paints a bleak picture: most of the traditional assurance methods are incompatible with Agile development approach. Noopur (2005, 19) supports this view by stating that nearly half of the traditional security assurance activities are not compatible with Agile Methods.

Security assurance method or technique		Match (2)	Indepen-dent (8)	(semi)-automated (4)	Mis-match (12)
Re-quire-ments	Guidelines		X		
	Specification analysis				X
	Review				X
Design	Application of specific architectural approaches		X		
	Use of secure design principles		X		
	Formal validation				X
	Informal validation				X
	Internal review	X			
	External review				X
Implementation	Informal correspondence analysis				X
	Requirements testing			X	
	Informal validation				X
	Formal validation				X
	Security testing			X	
	Vulnerability and penetration testing			X	
	Test depth analysis				X
	Security static analysis			X	
	High-level programming languages and tools		X		
	Adherence to implementation standards		X		
	Use of version control and change tracking		X		
	Change authorization				X
	Integration procedures		X		
	Use of product generation tools		X		
	Internal review	X			
	External review				X
	Security evaluation				X

Table 2: Compatibility analysis of software assurance methods (Beznosov & Kruchten, 2004).

4.2.2 A Dangerous Outcome: Vulnerable Software

This chapter presents the findings from the perspective of technical software security outcomes. The bottom line is that we need to understand what the weaknesses of our software are in different areas, why they manifest themselves and what are the real world impacts of those weaknesses. A limitation needs to be made clear at this point: the studies that are presented in this chapter have been performed against software that is produced by both Agile and traditional development methods. Therefore, it cannot be concluded that inadequate security has resulted because of either of those approaches. (Nor can we claim that either approach would be any better based on this data!)

Silverskin conducted a quantitative study about web application security in 2012. The study pulled together the results from audits of 130 customer applications. The tested applications included a wide range of different types: discussion forums, project management applications, web shops, data warehouses, issue tracking systems, healthcare services, sign-up services, web registration services, web banks, network games, gambling, etc.

According to the study (2012), human errors and development process deficiencies are the most usual causes of security problems. The most common sources of critical security deficiencies are lack of secure design and errors in implementation. Also configuration errors were found sometimes to be the reason causing serious security risks. The following categories of errors present the critical findings during different phases of development:

1. Design errors
    a. Are the hardest to correct and require a lot of effort to repair. The security vulnerabilities are a result of missing controls that would be necessary to prevent misuse of the application. The most alarming finding in the study was that important controls were missing entirely from the applications. (Figuratively speaking, they had built a house without remembering to buy lockable doors on the doorways). Namely, if access control mechanisms were found to be vulnerable, in more than 90% of those cases that mechanism was missing from the design altogether!

   b. **38 % of critical vulnerabilities** were caused by design errors.

2. Programming (implementation) errors
   a. Severe risks were often found to be caused by programming errors. Tiny programming errors could be enough to cause a serious data leakage. Luckily they are easier to correct than design errors.
   b. **47% of critical vulnerabilities** were caused by implementation errors.

3. Configuration errors
   a. The security of a well designed and implemented application can be ruined at the time of its installation and configuration. The platform that provides resources for the application (server hardware, operating system, application server, networking) may contain vulnerabilities that allow easy access to the whole system below the application. Configuration errors are the easiest to correct, but also the most prevalent in the study with more than 50% of findings. Luckily they rarely cause critical security risks.
   b. **16% of critical vulnerabilities** were caused by configuration errors.

Another way to categorize the results of the study is to present the critical security findings per control category failure (figure 8). This categorization shows that application input validation, session control and system protection are the most prevalent reasons for critical vulnerabilities.

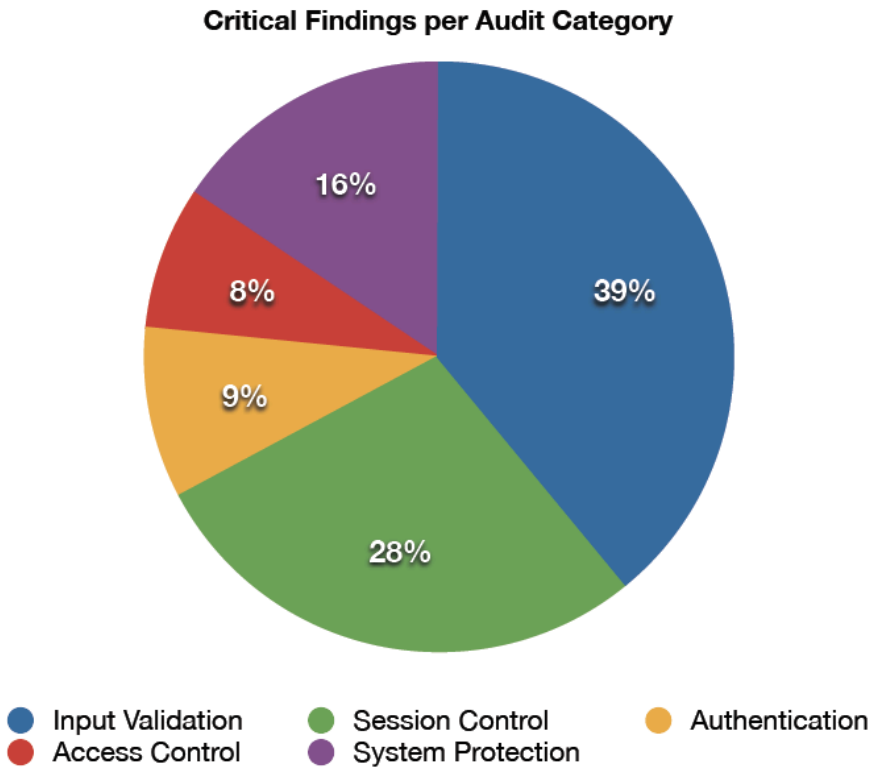


Figure 1: Critical findings per audit category (Silverskin, 2012)

Further analysis of the study findings shows the distribution of all vulnerabilities in the audit categories per origin (figure 9). It is interesting to notice that most input validation errors are due to implementation (programming) errors. Another interesting finding is that almost all access control vulnerabilities were caused by lack of design. It is worth mentioning as well that configuration-level errors are almost entirely caused by lack of system protection (figure 9). A question arises: perhaps the system protection and configuration should be included in test automation? And are the developers always skilled in system protection area, or do they simply perform the coding work with the application?

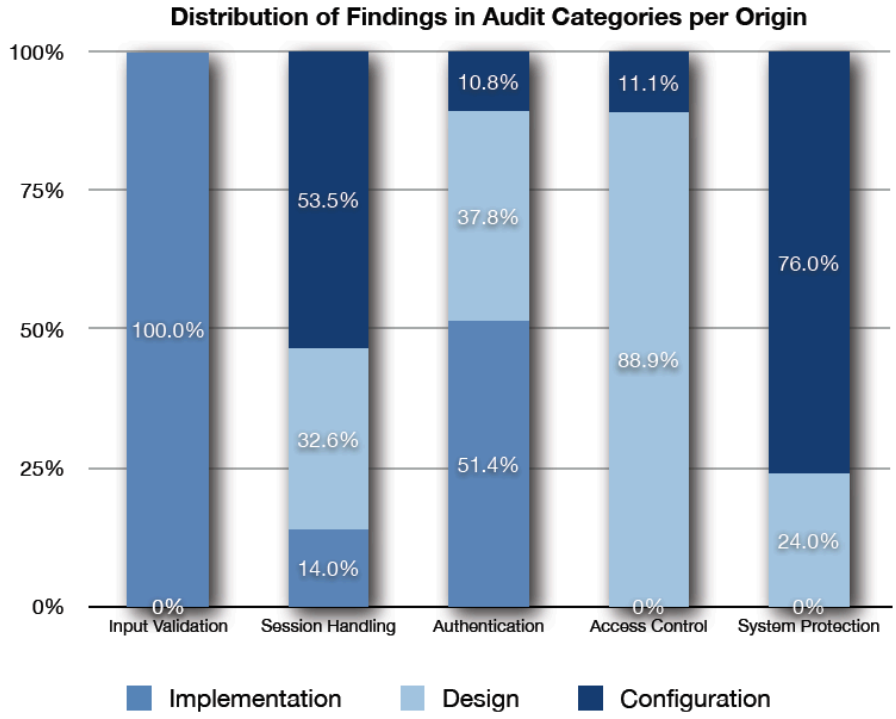


Figure 2: Distribution of findings in audit categories per origin (Silverskin, 2012)

Another source is WASC's Threat Classification Development View (WASC). This view presents different software security controls against design, implementation and deployment phases of development process. WASC explains that the view was created to illustrate "… where in the development lifecycle a particular type of vulnerability is likely to be introduced." It appears that WASC's recommendation is somewhat consistent with the results from Silverskin's study (figures 8 & 9 above).

Vulnerability	Design	Implementation	Deployment
Abuse of Functionality	X		
Application Misconfiguration		X	X
Brute Force	X	X	
Buffer Overflow		X	
Content Spoofing		X	
Credential/Session Prediction		X	
Cross-Site Scripting		X	
Cross-Site Request Forgery	X	X	
Denial of Service	X	X	
Directory Indexing			X
Format String		X	
HTTP Response Smuggling		X	
HTTP Response Splitting		X	
HTTP Request Smuggling		X	
HTTP Request Splitting		X	
Integer Overflows		X	
Improper Filesystem Permissions		X	X
Improper Input Handling		X	
Improper Output Handling		X	
Information Leakage	X	X	X
Insecure Indexing		X	X
Insufficient Anti-automation	X	X	
Insufficient Authentication	X	X	
Insufficient Authorization	X	X	
Insufficient Password Recovery	X	X	
Insufficient Process Validation	X	X	
Insufficient Session Expiration	X	X	X
Insufficient Transport Layer Protection	X	X	X
LDAP Injection		X	
Mail Command Injection		X	
Null Byte Injection		X	
OS Commanding		X	
Path Traversal		X	
Predictable Resource Location		X	X
Remote File Inclusion (RFI)		X	X
Routing Detour			X
Server Misconfiguration			X
Session Fixation		X	X
SQL Injection		X	
URL Redirector Abuse	X	X	
XPath Injection		X	
XML Attribute Blowup		X	
XML External Entities		X	
XML Entity Expansion		X	
XML Injection		X	
XQuery Injection		X	

Table 3: WASC view on development of threat classification.

Yet another interesting source of information is WASC's Web Hacking Incident Database (WHID). The database is " … dedicated to maintaining a list of web applications related security incidents" (WASC). The figure 10 below illustrates the real-world outcomes of insufficient web application security.

I am certain that not many people would be comfortable with the idea of enduring any of the top 5 impacts: hijacked user accounts, application downtime, information leakages, and malware being planted on our software or monetary loss. These top 5 constitute more than 80% of all impacts.

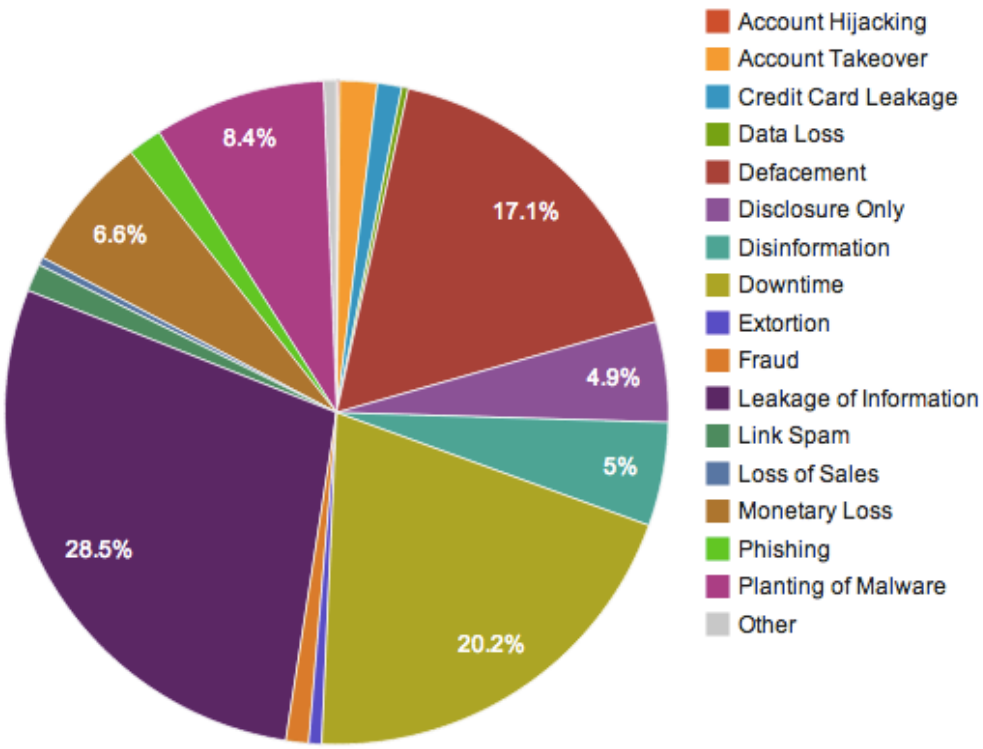


Figure 3: Outcomes of software vulnerabilities (WASC web hacking incident database).

The security outcomes in the figure 10 may seem alarming enough. Sadly there is even more reason to be concerned: WASC Web Application Security Statistics data (2008) also revealed that "…more than 13% of all reviewed sites can be compromised completely automatically".

4.3 The Benefits of an Agile Security Model

It should be evident that security in Agile software development requires a new, holistic approach to software security that is compatible with Agile Methods. Baskerville (2004) has also identified the necessity for such rapid and continuous development models. The goal of such a model could provide several benefits for many organizations and result in a more secure software. Benefits include, for example:

- Businesses which use Agile Methods could use the model for improving their software development processes
- Security service providers could establish new, innovative services for supporting their customers who are creating software using Agile Methods
- Software development industries could include effective factors from the model as a part of their industry standards and best practices
- Enable creating more secure software to the customers.

5 Research Process

The core theoretical framework in this work is based on Agile software development core methodology stated in the Agile Manifesto. Theories from traditional software development (e.g. waterfall) are also considered relevant to this thesis in the purpose of comparison and juxtaposition of the methods.
A theoretical framework is the generic angle from which the topic of this research has been approached. The framework explains the role of theories in the work.

The Agile Manifesto summarizes the common philosophy and approach shared by all Agile development Methods (flavors of Agile). At the highest level, the core values of Agile methodology are:

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
***Individuals and interactions** over processes and tools*
***Working software** over comprehensive documentation*
***Customer collaboration** over contract negotiation*
***Responding to change** over following a plan*
That is, while there is value in the items on the right, we value the items on the left more" (agilemanifesto.org).

It should be noted that the four values are clearly chosen to display the distinction against traditional software development methodologies (e.g. juxtaposition of the working software against comprehensive documentation. Comprehensive documentation is considered to be a requirement for traditional software development).

The Manifesto is further broken down into twelve core principles:

- "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.

- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly." (agilemanifesto.org)

In this research, a risk analysis is performed that uses the Agile manifesto's principles as a framework. Ensuing, an Agile security model is formulated based on literature and interviews of Agile security experts. Finally, reflections on real-life customer cases are used to test the security model's applicability.

5.1 Constructive Research Approach

This research uses a constructive research approach. The research is performed with the intention of improving the business processes and services of Silverskin Information Security LLC, and for the benefit of the security community and Agile software development organizations. My intention is to achieve this goal by constructing a novel Agile Security Model that combines theories and experience from several fields of study such as Agile software development, security engineering and security management.

5.1.1 Philosophy and Theory of Constructivism

In philosophy, constructivism is a view in which all knowledge is considered to be a compilation of human-made constructions. The opposing philosophical view of constructivism is objectivism where objective truth is discovered via neutral and impartial discovery. In constructive approach the paradigm of knowledge is shifted from objectivity into inter-subjectivity. This inter-subjective approach has implications on the definition of truth. Instead of objective truth the focus is placed on viability of artefacts, or constructs. As Vico said: ”The norm of the truth is to have made it” (Wikipedia). Another significant characteristic of constructive research is that constructions are not discovered. The constructs are invented and developed instead. The development of the construct creates something entirely new and profound (Lukka, 2011).

According to Crnkovic, the key idea of constructive research (also known as constructivist knowledge production) is to create a construction that is based on some existing knowledge by adding new links and interconnections to the knowledge. The creation of the construct proceeds by the means of design thinking: a vision or a projection of future artefact/theory is envisaged which fills conceptual and knowledge gaps by building purposeful blocks to support the whole construction (Crnkovic, 2010).

Typical artefacts of constructive research are ” …models, diagrams, plans, organization charts, system designs, algorithms, artificial languages and software development methods are typical constructs used in research and engineering…” (Crnkovic, 2010, p. 2). Constructivist artefacts are usually the result of design and development instead of discovery (from nature). A construction constitutes a new explanation for reality that can be examined and understood against the existing ones.

Constructive research approach emphasizes the practical aspects of the research process. The approach requires that experiences are obtained from the problems and solutions from the interactions of real world. The role of the researcher in this approach is to participate in the everyday activities, service

development and innovations of the company that supports the research (Virtanen 2006, 47-48). From the perspective of the researcher it is essential, in the analysis phase of the research, to maintain critical attitude towards the constructed artefact and the organization (Lukka & Tuomela 1998, 24-25).

In summary, the following are key characteristics of constructive research approach (Lukka, 2001 & Anttila, 2007):

- Is based on problem of real world.
- There is a clear justification, a need, to solve the problem.
- Creates a novel and innovative construction to solve the problem.
- The developed construct is tested or applied in practice.
- Involves learning from experience in teamwork between the researcher and the people who perform the work in practice.
- Is based on existing theories.
- Also considers how new empirical findings could be interpreted from theoretical perspective.

Basically, constructive research is experimental, i.e. the developed construct is tested in practice. Under ideal conditions a real world problem is solved by the new construction. At the same time, the research process creates new theoretical insights. Also failed projects may be beneficial to the research because the reasons for their failure can be analyzed. It is always also possible that the failure of projects could have been avoided (Lukka 2001).

5.1.2 Constituents of the Constructed Agile Security Model

The Agile security model in this research was conceived, modified and finally documented during various customer projects of Silverskin. Agile development literature and interviews of Agile Experts were used as other sources of information. Theoretical aspect of the research (literature & interviews) provided the framework that supports our observations and experiences. The figure 2 illustrates the constituents of the Agile security model.

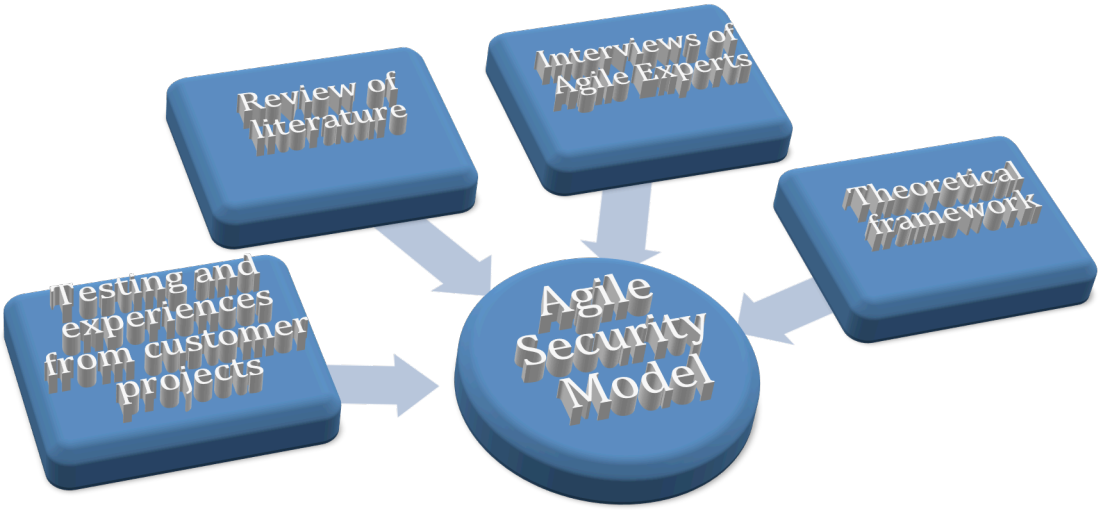


Figure 4: Constituents of the constructed Agile security model.

The following table shows the different types of customer projects and the phases of software development where I have interacted with Silverskin's customers. It should be noted that some customer projects covered various phases of software development although the table 1 only refers to one phase per project type. The same project therefore may appear under several project phases. (This does not create a problem because we do not need to analyze the number of conducted projects in this research.)

Types of customer projects where we tested parts of the security model in this research:	**Software development project phase**
Conception of the security-centric system and business case development, high level user story development and related preliminary feasibility studies	Initial planning
Application and system threat analyses and related user story and attacker story development (use cases and misuse cases)	Requirements

Providing security awareness training for developers and testers	Requirements
Defining security requirements for software projects	Requirements
Design of secure software, system and network architectures	Design
Code reviews and pair programming	Implementation
Testing of vulnerabilities, penetration testing, ethical hacking, fuzzing, performance testing and test automation	Testing & Deployment
Documenting security features and mechanisms (security compliance)	Design, implementation & Deployment
Performing various types of acceptance and compliance audits, both technical and administrative	Implementation & deployment

Table 4: Silverskin's undertakings where the constructed security model was tested.

The above engagements with Silverskin's customers are one significant source that constitutes practically tested key building blocks in the Agile security model.

5.2 Phases of Construction

Figure 5: Construction phases of the Agile Security Model

The first phase of creating the Agile security model was to establish a goal with Silverskin in order to research the subject. I have been actively involved in many of the customer cases where Silverskin has provided professional services for its Agile customers. My role in most projects has been to work as an internal advisor or peer reviewer. In those instances I provided new ideas, feedback and helped others to reflect on our progress. In some projects I have been the primary security expert who provided the service. Agile Expert C has been especially active in executing the consultancy services.

In the second phase of this research, a rather large number of different customer projects were executed during a 2-year period. The projects consisted of various assignments types that are enlisted in the figure 3 above. Numerous parts of the Agile Security Model were tested in practice during these assignments. We executed our projects and simultaneously learned more about solving their security problems in practice. At the same time, theories about secure Agile development were used in various phases of the project in order to reflect on our performance.

Not all of the abovementioned customer projects were strictly related to Agile (i.e. some customers seemed to use waterfall approaches of software development). Nevertheless, the service that we provided was applicable to, and part of the holistic Agile security model. For instance, a penetration test against a web application is agnostic to the development methodology per se because it only measures the behavior of the software.

The development of the model was an iterative process where our best understanding of the subject was tried, tested and improved in each phase of the customer engagements. There simply is not any generic formula or method for this phase of development where the model was tested, applied and modified until its constituent components fit well in the model. I was personally actively involved in most of the customer projects where we applied the model, or parts of it. As the iterative process progressed, we could analyze and reflect

in practice what worked and what did not. Lukka & Tuomela (1998, p. 25) state that usually constructive research progresses iteratively to practical testing of the construct. They also state that unless the model is tested in practice the research process should not be considered successful.

Finally I decided that our experience from testing the model in customer projects had progressed enough to be documented. The process of testing, trial and iterative development had taken about two years at this point (from fall of 2011 to August 2013). It was time to take a step back and pull the pieces together, analyze the results and document the model.

5.3 Research Methods and Data Collection

Main data collection method in this research is based on Interviews of leading Agile security Experts. General understanding about the topic of secure Agile software development was first obtained by reviewing and analyzing literature such as scholarly articles, books, standards and research papers.

5.3.1 Review and Analysis of Literature

First a review had to be conducted into the available literature in secure Agile software development. The following sources were used to search for suitable sources, mostly books in e-format:

- Laurea's electronic Metalib–library
- Google Scholar

Meta–results (keywords) and tables of content have been inspected in order to identify those sources that contained relevant information about the subject of this research. After reviewing the e-books it soon became clear that the literature contained scattered and apparently incomplete results about the subject. Not a single source could explain and provide a holistic model about

secure Agile software development. Therefore the search was widened to cover other relevant sources such as:

- Research papers about security in Agile (mainly Google Scholar)
- Industry best practices from world's largest IT companies who use Agile software development approaches (Microsoft, Cisco, etc.)

Sixteen different sources containing e-books, research papers, standards and company-specific development methods were collected and analyzed. The analysis consisted of reading the sources and dissecting their contents on a Microsoft Excel worksheet. Whenever a source suggested that a security mechanism should be in place, the proposed mechanism was added on the worksheet (table 2). Result of the literature analysis was a long collection of Agile security techniques that were proposed by various sources. Some of the proposed techniques were common across almost all of the sources. There seemed to be wide agreement about their usefulness. For example, conducting some sort of an application threat analysis was common to most sources. At the other end of the spectrum, a large number of proposed techniques seemed to appear only in a few sources.

Proposal	**Source 1 E-book**	**Source 2 Research paper**	**Source 3 Standard**	**… 16 sources …**	**Total # of proposals**
Formulate functional security requirements	1		1		2
Conduct threat assessment	1	1	1	1	4
Code review			1		1
…	…	…	…	…	…

Table 5: Literature analysis technique.

As some security techniques seemed to appear more often than others, a presumption was made that popular techniques are generally considered to be reliable and more effective than those techniques that were mentioned only

in a few or in just one source. This research partly relies on this presumption in proposing certain security mechanisms while excluding others. Confirmation of this presumption was obtained in the interviews of Agile security experts. During the interviews it was important to avoid confirmation bias. Therefore, structured open interview technique was used.

The above literature analysis method can be criticized by questioning whether the most common security techniques are always effective in all customer cases. This critique is valid because based on the used analysis technique it is not possible to state that a security technique which was mentioned solely once could not prove to be useful in an arbitrary single case somewhere in the world.

However, during the interviews, Agile security expert A did comment on this limitation with a rather humoristic comment: "… but it is (selection of security techniques) based on the idea that a million flies cannot be wrong. i.e. if several large companies who operate in market economy have decided that these things are necessary and beneficial they probably have sound reasoning behind it. If you, as an individual, are sure that these security techniques are not worthwhile in your case then your business might be a little different from others. But this is rarely the case. It is commonly known that all snowflakes are individual and different. But if you take a step back and watch them from the distance they all look the same!"

5.4 Critique of Normative Approaches in Software Security Development in Literature

A critique towards normative approaches towards security is necessary since those (prescriptive) approaches to software security are common (for example, VAHTI 3/2013, ISO/IEC 27000 series and other standards).

During the literature review it became apparent that certain criticism is appropriate towards certain approaches prevalent in Secure software develop-

ment literature. The approaches in literature can be characterized as either process-oriented, i.e. they are based on methodological or organizational approaches like Microsoft's), or normative-oriented (prescriptive). A couple of contrasting examples are in order:

1. VAHTI 03/2013 is a recently published Finnish secure software development guideline that sets requirements for organizations in the public sector.
2. Extending XP practices to Support Security Requirements Engineering (Beznosov, Boden, Boström, Kruchten & Wäyrynen, 2006) is a research paper that proposes several Agile process–oriented security controls.

The following table 3 compares these two sources against each other:

Source	# of proposed mechanisms	Type of proposed controls	Approach	Type of document
VAHTI 03 / 2013	82	Exogenous (external)	Prescriptive	Governmental guideline
Extending XP …	12	Endogenous (internal)	Process / organizational	Research paper

Table 6: Comparison of two different software development literature sources.

Comparison of these two sources juxtaposes the two general approaches that authors propose for gaining control over software development security.

Critique of prescriptive approaches in literature

The word "prescriptive" is etymologically related to the same word that we use when a medical doctor prescribes a medicine to a patient. In prescriptive software development approaches (i.e. VAHTI 03/2013) gaining control over software security is achieved by proposing a wide number of activities and security mechanisms. The tone and the level in which such requirements are written usually state that the requirements should preferably be implemented in all cases where software is developed without exception. These security requirements are written on detailed level of single controls rather than on

level of processes or methods. For example, a prescriptive control could state that: “the organization shall deploy a penetration test against the product, using OWASP top 10 categorization. The test shall be performed by an independent third party.” This prescription of a detailed security task is given without any knowledge of the product or requirements of the customer case.

Theoretically therefore the prescriptive approach is like a blind and deaf doctor who is unable to perceive and accurately diagnose his patient’s illness. The patient trusts the good doctor and expects to receive help for his ailment (so do our customers who place their trust on us, the security practitioners). The impaired doctor is well educated and licensed to perform his profession but he is unfortunately unable to directly perceive, analyze or adapt to the customer’s acute needs since he is deaf and blind. The good doctor, hoping to cure the customer’s ailment, resorts to prescribing all types of available medicines (security controls) that he has observed to be useful in various cases before he became limited by his deafness and blindness. As a result, the number of different medicines that the customer needs to ingest is astounding. Unluckily there is no guarantee that any of the prescribed medicines are effective against the customer’s illness. To aggravate matters, some of the medicine may prove to be harmful to the customer’s health. The approach is irrational because the prescriptions are not viable from economical viewpoint. There are an infinite number of possible diseases in the world. Eventually this approach forces the patient to find a doctor with an unimpaired vision and hearing.

A serious question arises: do the prescriptive approaches suffer from a lack of holistic framework and adaptation mechanisms that could make them applicable in variable cases of illnesses? It might be that each prescriptive security requirement may actually have some amount of empirical evidence to support it in some instances. The medicine may work if there happens to be just the correct disease at hand. In other words, security practitioners may have really proven that each prescriptive requirement is good for the patient in some occurrences. This is evident and therefore it should not be thought that any sin-

gle requirement in such a prescriptive source (such as VAHTI 3/2013) is universally flawed.

Any single security control is not criticized per se, and this is not the point of this critique. The prescriptive approaches emphasize the question of "**what** security mechanisms should be implemented?" An entirely different question should be asked altogether right from the outset. We should ask ourselves: "**how** should we integrate security engineering in the Agile methodology, and **in what manner** should security mechanisms be applied to it?" Most of the prescriptive approaches have failed to ask this proper question in the first place. Diagnosis of the problems must preclude the prescription.

5.5 Interviews of Leading Agile Security Experts

Three interviews were conducted in order to collect expert knowledge about secure Agile software development. The interviewees are all experts who have been working in the field of software security and Agile software development for several years. All of the interviewees are persons who are well known and enjoy wide respect of the Agile security community in Finland.

The interviews were conducted on following dates:

- 1st of March 2013 – Agile Security Expert C
- 2nd of July 2013 – Agile Security Experts A and B (separate interviews)
- 14th of October 2013 – Agile Security Expert C

The interviewed Agile experts shall remain anonymous because confidentiality of their personal information needs to protected. The identity and each interview of the experts is documented in the research documentation and retained by the author.

It was necessary select people in the interview sample who had adequate experience from Agile software security. Another considered factor in the selec-

tion was that the interviewees shouldn't work in similar positions and they shouldn't work in the same company.

The Agile experts had the following professional qualifications and characteristics:

Agile Expert A: works in a Finnish security-centric software development enterprise. Expert A's work consists of directing Agile development security in his company. He has worked in software R&D for 17 years, of which four years in an Agile environment, and 10 years in software and product security. Expert A has published and spoken on the topic of agile security both locally and internationally.

Agile Expert B: works currently as a CEO in a well known Agile software development company in Finland. He has a background in security auditing, testing and consulting businesses. His experience covers a wide variety of different positions, including sales, marketing, general management and product / service development in various companies. He has experience in buying, selling and developing security testing services and Agile software projects. He has obtained experience from almost all commercial aspects around the subject of this research. Expert B was selected to be interviewed because:

- he has deep understanding and experience with customers who want to purchase (secure) software from Agile development companies
- he understands what kinds of commercial security services are currently available in the market in Agile software development markets
- he comprehends the meaning and impact of security to the Agile development from the financial viewpoint better than any other interviewee. Having this insight in the scope of this research is irreplaceable since literal sources had almost nonexistent information about the financial aspect of Agile security investment.

Agile Expert C: works in Silverskin Information Security as a senior software security advisor. He has conducted scientific research about security of web

applications and secure Agile development. He has talked in international conferences about software security (for example SANS institute's conferences). He also provides training to Agile developers about secure application and system development, penetration testing and ethical hacking. He also conducts system and application level penetration testing and audits on a daily basis in his work. His qualifications also include numerous internationally accepted technical and administrative security certifications in his field of expertise. Expert C was selected to be interviewed because:

- he has years of experience from wide variety of security–related specialist tasks, especially from security testing, penetration testing and application security domains.
- he has created and published research papers about application security and software security
- he has consulted, trained and taught countless Agile software developers to improve themselves in software security

Finally, after having done three interviews, a decision was made that no more interviews needed to be conducted because a certain informational saturation point had been achieved. At the end of each interview, the interviewees seemed to repeat the same issues that other interviewees had already stated. It was assumed that not much new insight can be gained from further interviews.

The interviews were conducted as open interviews where the Agile Experts were allowed to talk freely about Agile development security. Interview questions were not prepared upfront. The interviews were directed by asking open questions about different phases of Agile development process (inception – requirements – design – implementation – testing – deployment) and in what manner security should be optimally handled in each of these phases.

At the time of interviews I had already read many books and articles about Agile security. Since I no longer had the mentality of a beginner, there was a slight risk of confirmation bias. The open interview technique was chosen to

specifically counter this bias. Confirmation bias means that personal opinion or knowledge possessed by the interviewer influences how interviews are conducted and their results are interpreted.

6 Traditional vs. Agile Software Development Approaches

De Lucia & al. (2008, p.249-251) note that traditional software development methods such as waterfall and spiral methods necessitate a good knowledge of the specific application and the requirements and needs of the customer. They also acknowledge that these techniques mitigate the uncertainties of software development by using very detailed up-front plans. In effect, every feature and quality of the end result is specified at the beginning in order to avoid expensive changes late in the project. They also criticize the approach because "in certain application domains and for certain problems, plans simply do not work or they are inefficient. This is true regardless of the quality of the people involved in the project. Even if plans are supposed to help organizations, many project managers acknowledge that in many kinds of projects they cannot follow them due to market needs…"
According to Boehm (1983), the waterfall method proceeds linearly through the following phases: feasibility, plans & requirements, design, programming, integration & test, maintenance and phaseout.

6.1 Agile Software Development

Often Agile is feared to be a sort of an uncontrolled software development show of autonomous rock stars. This is sometimes called a "cowboy Agile", or even "a hippie anarchist plot" (Vähä-Sipilä, 2011). According to Beznosov & Kruchten (2004) Agile is based on less formalized, often less visible and softer control mechanisms than traditional software development: "…all agile methods exhibit a great aversion for software bureaucracy, and favour direct communication between participants rather than reliance on written arti-

facts." We need to differentiate our understanding of Agile software development from this perception and see what Agile is in reality.

Agile Methods (AM) are a set of software development methods and techniques. They have been conceived to deliver their results timely, on budget, and with high customer satisfaction. According to Strode (2006), the most popular Agile Methods are:

- Dynamic Systems Development Method (DSDM)
- The Crystal methods (Crystal)
- RUP (dX)
- eXtreme programming (XP)
- Adaptive Software Development (ASD)
- Scrum (Scrum)
- Pragmatic Programming (PP)
- Internet Speed Development (ISD)
- Feature Driven Development (FDD)
- Open Source Software Development (OSS)
- Lean Development (LD)

The roots of Agile Methods are based on application of principles from lean production that were developed in 1950s by Toyota, and since then have been taken into use in most manufacturing processes world-wide. The leading idea in lean production is that anything that does not add value to the customer is a waste and should be constantly identified and removed (De Lucia & al. 2008, p.250).

Agile software development Methods have been born out of necessity to answer the problems of deterministic (waterfall) methods, especially projects going over budget and schedule, or just simply failing to deliver (De Lucia & al. 2008, p.249 and Beznosov, & Kruchten, 2004). Specifically, Agile Methods (AM) simply help developers to focus on the objectives of their customers and help delivering the products without wasting effort on issues that do not gen-

erate value to the customer. As opposed to the traditional software development models, AM does not require initially deep knowledge of the requirements of the customer or the end users. In AM it is admitted that

- precise understanding of all customer needs is not practical,
- requirements are not stable, the customers may even change their requirements during the development process,
- a priori specification of requirements in a complete way is not possible,
- some software features are irreversible, or hard to change after they are implemented without seriously impacting the scheduling and the budget of the project.
- often the customers are not even able to specify all required main functionalities of the applications. How could security be defined upfront in such a fuzzy environment?

6.2 Comparison of Control Mechanisms in Agile and Traditional Approaches

According to De Lucia & al. (2008, p. 264) all production processes are regulated by some kinds of control mechanisms. The control mechanisms are a means to ensure that a common goal has been achieved. This view interdepency between the activity and resources has been also noted by Malone & Crawston (1994). Controlling the resources that actors depend on endogenous control over the process. There are two types of control mechanisms (table 4):

1. Exogenous (or external control) that adds rules to the process
2. Endogenous (or internal control) that defines the control rules as integrated part of the process

Type of control	Use of control rules	Separation of control from process	SW development	Control type: Detective Preventative Reactive
Exogenous	Added on top of the process	Possible	Traditional approaches	Detective, Corrective
Endogenous	Integrated into the process	Not possible	Agile approaches	Detective, Preventative

Table 7: Analysis of internal and external process controls.

Exogenous control adds rules 'on top of' the process, which implies that the rules can be separated from the process without intercepting the process flow. This is common in traditional software development approaches where security requirements can be identified and written separately from all other phases of development. For example, the security requirements could be identified and documented but not mandatorily included in any subsequent designs, implementation or testing of the software product. This sort of control is rather detective or corrective; i.e. the customer is able to detect that security mechanisms are necessary and should be applied. If he detects they were not implemented, he may require the observed discrepancy to be corrected. In exogenous control the process does not prevent per se skipping the implementation of security. Sometimes the skipping of necessary security mechanisms may go totally undetected. "We wrote the paper but simply did not do it in practice". In such case the customer may have had very detailed documentation in place but the reality of software development simply did not agree with the documentation. In such cases the documentation is not in accordance with the real world status of the software product.

Agile security expert B stated in his interview when asked about the differences between Agile and traditional development Methods that "…I think documentation in Agile is closer to the reality how real software development works. And waterfall model is some kind of a lady's ideal state (of documen-

tation).” He also explained that ”… basic idea is that in waterfall model, the documentation (requirements) is assumed to be universally correct but software is considered to be broken if there is a discrepancy between them…” and contrasted this with Agile: ”…in Agile, the documentation is either up-to-date or it is not. But if a paper document states something and that something is not implemented in software, then the documentation should be considered broken, not the software.” The expert B’s opinion reflects a preference towards endogenous (internal) process control instead of exogenous control through precise documentation.

In contrast to exogenous control, endogenous or internal control includes the rules as a part of the process itself. The process itself is designed so that control mechanisms are embedded into it. Separation of the control mechanism from the process is not possible because removing the control would disrupt the process itself or render it inoperable. Malone & Crawston (1994) propose several meachanisms for coordinating and conrolling activities in various circumstances. They explain the endogenous control mechanisms that are used in Agile, such as:

- Shared resource management
- Managing (customer) relationships
- Managing simultaneity constraints
- Managing task / subtask relationship

According to De Lucia & al. (2008) also comment on these endogenous mechanisms: ”…Agile Methods are designed to force developers to coordinate without asking them to do it explicitly, limiting the unproductive activities needed only for coordination.” They also propose that endogenous control yields the benefit of preventing the skipping of relevant security mechanisms and detects if they are not properly implemented, provided they are detected in the first place. The endogenous control prevents the process from working if a problem is not solved. For example, if pre-defined quality criteria are not met, a task will not proceed. A good example of an endogenous control in Agile is the following: ”if a software build fails, all other work is stopped until the problem is solved” (Agile Expert A). Another example would be Malone &

Crowston's (1994) task / subtask relationship control: A product backlog is decomposed to smaller doable items and those tasks are added to a sprint backlog. Explanation of generic backlog management in Agile follows later in this research.

For the above reasons, endogenous control can be considered "strong" and exogenous "weak". Malone & Crawston (1994) characterize this difference by giving the following definition for coordination: "Coordination is managing dependencies between activities." We use the word control in the same context and meaning in this research. Coordination of dependencies gives endogenous control over activities.

According to Sillitti & Succi (2008) the Agile Manifesto applies the tenet of endogenous control and uses it all over its principles. Their example of endogenous versus exogenous controls in extreme programming is presented in the table below:

Practice	Control
Planning game	Endogenous
Short releases	Endogenous
Metaphor	Exogenous
Simple design	Exogenous
Test-driven development	Endogenous
Refactoring	Endogenous
Pair programming	Endogenous
Collective code ownership	Endogenous
Continuous integration	Endogenous
40 hours week	Exogenous
On-site customer	Exogenous
Coding standards	Exogenous

Table 8: Exogenous and endogenous controls in extreme programming (Sillitti & Succi, 2008)

According to Malone & Crowston (1990), two resources can be dependent only via some kind of a shared common resource. They proposed that there are three types of dependence: sequential, shared and common (table 6).

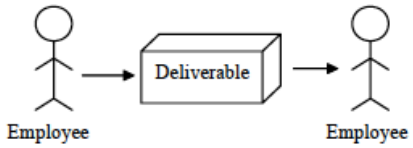
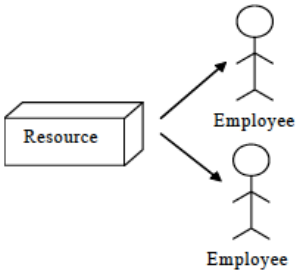
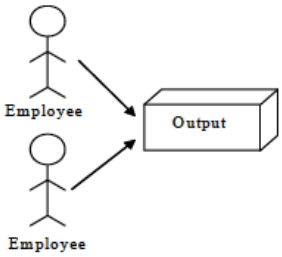
Sequential control	Shared resource	Common output
		

Table 9: Sequential, shared resource and common output controls (Malone & Crawston, 1990)

Malone & Crowston (1990) provide the following explanations for the control types:

1. Sequential control: is achieved when a task creates a resource or output that another task requires as an input. There is a dependency between the tasks that requires correct order of execution. According to Vähä-Sipilä (2011) sequential control mechanisms are not always suitable in Agile because they easily create technical queues which tend to decrease productivity. One practical example of sequential control in Agile could be the collection of user stories by the product owner and importing these stories to product backlog. The sprint backlog can be formulated based on the most important stories on the product backlog. This produces a sequential control queue.
2. Shared resource: control is achieved when multiple tasks require a shared limited resource. A sprint or product level backlog is a good example of resource dependency. All developers in the team are dependent on the backlog for prioritization and selection of their work tasks. For example any task on the sprint backlog has a good chance of being done.

3. Common output: control is achieved when several tasks contribute to create the same output. There is an inherent risk of duplicate work or waste of resources. Ideally, the two tasks should contribute two different aspects of a common resource to avoid overlap. This is a common problem with several actors who collaborate to achieve a common goal. A good example of common output would be creating a security architecture design in collaboration with the development team members.

6.3 Sequence of Software Development Phases

Sillitti & Succi (2008, p.12) state that traditional software development methods share common sequential development phases of analysis, design, coding and testing. This generic process is illustrated in the figure 10 below. It should be noted that the picture does not take into consideration the initial planning and deployment phases of software development. Nevertheless, the linear nature of the development process is apparent.

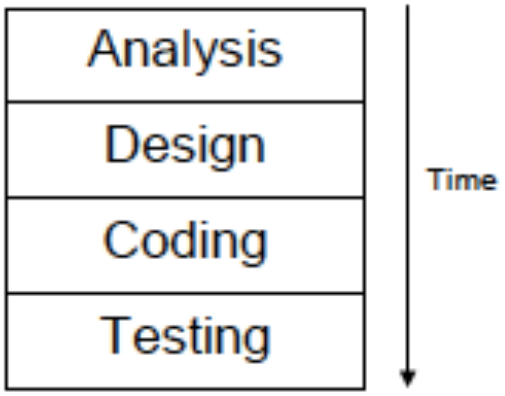


Figure 6: Traditional waterfall software development model (Sillitti & Succi, 2008)

According to Sillitti & Succi (2008, p.12) in Agile, the development process is iterative and organized differently. The formal pre-defined phase does not

exist anymore. During every round of the process the developers need to produce something valuable for the customer. For every user story requirement, developers do a little bit of analysis, design, testing and coding. Interesting points in the development arise: tests are sometimes developed prior to code and analysis and design happen across the entire development process.

The figure 5 illustrates the iterative nature of Agile approach. The phases of analysis, design, testing and coding overlap with each other and are repeated many times in consequent iterations.

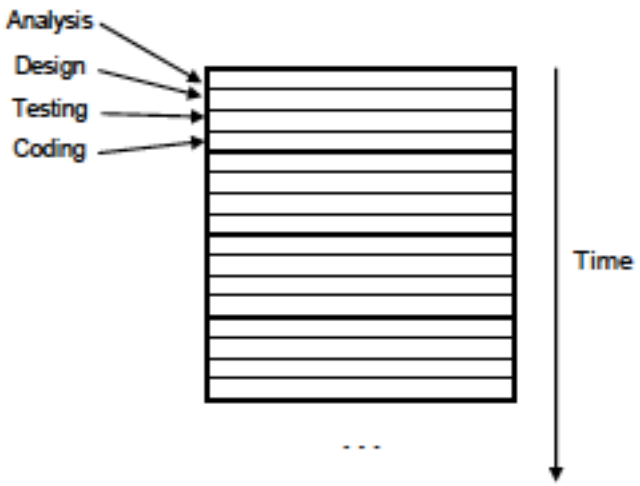


Figure 7: Iterative Agile software development process model (Sillitti & Succi, 2008)

The iterative development cycles of an Agile development project are managed by using a mechanism called “a backlog”, or often called a “sprint backlog”. The backlog contains a list of the tasks and requirements to be completed within one development cycle (sprint). A higher level “product backlog” usually exists independently of the sprint backlog. The prioritization of items on the product backlog defines which tasks shall be included in the following sprints (and sprint backlog). The product backlog is basically a similar performance list with slightly superior items on it. The product and sprint backlogs are one of the core control mechanisms in Agile (especially in Scrum method). According to Layton (2013), the sprint backlog includes:

1. List of user stories, in order of priority for the current sprint (cycle)
2. The effort estimation per story
3. The task definitions for each story
4. The effort estimation in hours for completion of each task

The development team itself is responsible collectively for creating and maintaining the sprint backlog. The backlog is in fact, a modern snapshot of the progress of the current sprint.

One benefit of Agile Methods is their adaptive nature. According to Vähä-Sipilä (2011), when business targets change radically late in the software development, this may force the developers to a 'pivot' movement in the development. Earlier software development progress may need to be scrapped, or wasted and new functionalities implemented. By pivoting in Agile you may save these software increments that can be reused. With the waterfall, one would probably have to throw all of the earlier progress away. In some cases this could save the company (figure 6).

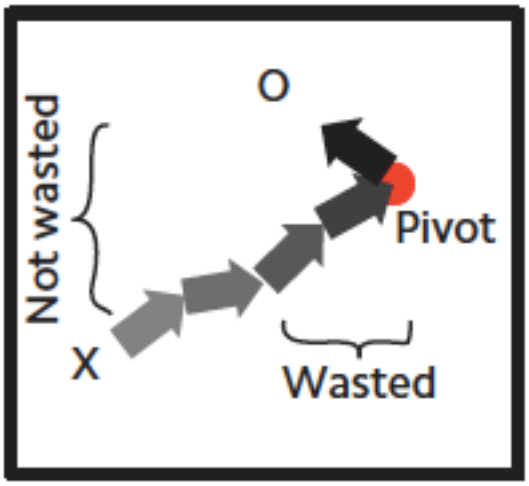


Figure 8: Agile pivot when business targets change (Vähä-Sipilä, 2011)

6.4 Short Explanation of Scrum Method

The scrum is a popular flavor of Agile Methods. Vähä-Sipilä (2011) characterizes the Scrum with the following five features:

- A product owner (PO) is constantly scouting the business environment and trying to guess where the goals of the development are (the letter O in Vähä-Sipilä's picture N)
- The PO constantly maintains a prioritized list of appointments called the 'product backlog'. Things up on the list are the most important ones, and down on the bottom are the ones that may never get executed. The matter of prioritization is a business decision.
- The increments in the software development process are called 'sprints'. Each spring lasts for about 2-4 weeks. At the start of the sprint, the development team takes the most important items from the top of the product backlog and takes them to a 'sprint backlog'.
- At the end of each sprint, the development team has implemented, tested and deployed the items on the current sprint backlog. A sprint review is then conducted, where the team decides if the work is free from 'technical debt', and whether the work is 'done' or not. If everything works smoothly, the items on the product backlog can be accepted to be finished and next sprint may begin. During the sprint, the PO may have shifted the priority of the items on the product backlog, thus changing the direction towards the business target.
- The development team's scrum master is a person who ensures the team's well being. His responsibility is to remove any obstacles that may obstruct the work of the development team.

According to Vähä-Sipilä, Agile product management works by identifying business-level targets, and then churns them into smaller pieces until the tasks can be imported and prioritized on the product backlog (figure 7). Business-level requirements are sometimes called 'epics' that are 'decomposed'

and prioritized into smaller items on the product backlog. The process of decomposition is in fact central sequential control mechanism in Agile.

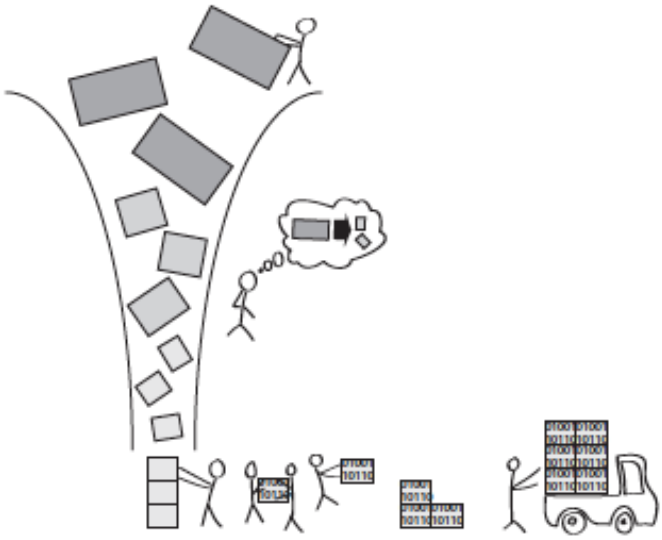


Figure 9: Agile product management funnel (Vähä-Sipilä, 2011)

7 Analysis of Security Problems and Applicable Security Mechanisms

All Agile Methods are based on the core principles stated in the Agile Manifesto. The problem is that the Agile software development methods are perceived to lack adequate means for providing security assurance (Beznosov, 2004 & Baskerville, 2002).

Therefore we are going to analyze the Agile core principles by identifying subproblems associated with the principles, their causes and their probable impacts. The terms ”perceived problem” and corresponding ”impact” are used to describe the identified problems of Agile principles that could lead to unwanted outcomes. In a sense, this analysis could well be interpreted as a sort of a qualitative risk assessment.

The term ”perceived problem” is used here in a qualitative manner because our problem space consists of various interlinked factors that are not easily

transformed into numeric format. For instance, how would one value the security knowledge of one's developers, or the risk of not having adequate knowledge?

The problem-based construction process follows the following steps (Figure 11):

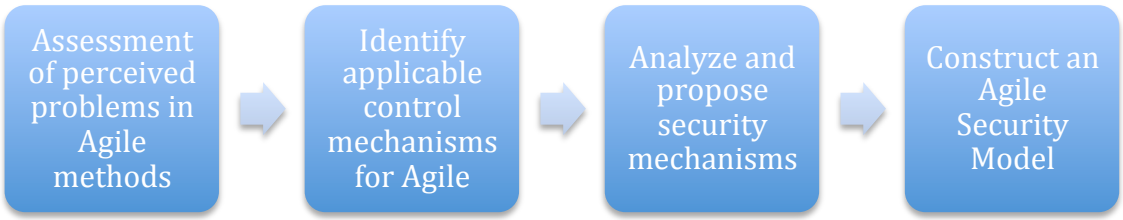


Figure 10: Problem-based construction process for the Agile Security Model.

Analysis of the perceived risks, impacts and Agile core principles is presented in the following tables.

Agile core principles	Perceived problems	Impacts
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. Businessmen and developers must work together daily throughout the project.	Identifying relevant security features and suitable level of security for the customer is insufficient.	If the security features are not identified correctly early in the development process, it may be too late to implement them later on. Conversely, too much and too expensive security might be implemented.
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. The best architectures, requirements, and designs emerge from self-organizing teams.	How to design security –related software features in a way that allows flexibility even in the event of radical refactoring? How to establish change control?	Rigid security architecture may be a reason why application cannot be flexibly refactored. This would lead to loss of value to customer. Uncontrolled changes can break security architecture.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. Continuous attention to technical excellence and good design enhances agility. Simplicity--the art of maximizing the amount of work not done--is essential. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	How to identify, design and implement the right security features at the right time in the development process when they are needed?	Security features and tasks may remain as a amorphous conglomeration of unmanaged stories that is never properly designed, implemented nor finished. This creates technological debt. Also, totally ineffective or incompatible security features may be implemented.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	How to manage the human aspects of secure Agile development, the security awareness aspect?	Lack of security awareness may prevent any security activity from taking place. At minimum, it will slow down progress.
Working software is the primary measure of progress.	How to ensure that software is not infested with security vulnerabilities after the point where software is deemed to work?	If software vulnerabilities are not managed during the development, they are bound to emerge in the software product.

Table 10: Core principles, problems and possible impacts in Agile software development.

7.1 Identification of Applicable Security Mechanisms

An analysis of relevant literature was necessary to clarify which security mechanisms (or security recommendations) should an organization include in its Agile software development model in order to create secure software. We need to understand from a holistic perspective which security mechanisms seem to be the most relevant. Also Siponen & al. (2005) have identified the need to find security approaches that are adaptive to Agile Methods, instead of imposing external security mechanisms to development.

While analyzing the Agile literature the following observation became apparent: There doesn't seem to be agreement on which security mechanisms are critical, or even remotely relevant to Agile, and which have a less observable impact, or no impact at all. Furthermore, most of the sources did not contain any kind of references or analysis of sources of information, and did not attempt to clarify which recommendations are the most relevant. Because of this, it was necessary to conduct a meta-level analysis about the relevance of various security mechanisms before any further conclusions are drawn. The question is: "which recommendations (security mechanisms) are commonly considered to be necessary in most cases, and which ones are not?"

Since Agile development Methods do not fit well with traditional, normative security requirement frameworks (Baskerville 2002 p.337-346, Beznosov 2004 and Särs), it is an imperative to identify the factors on which most authors agree. We should also strive to keep the number of our recommendations on minimum for the sake of efficiency. Only such security tasks that arguably do have an impact on the end results of the SW development should be recommended. I strongly believe that cluttering the software development process with ineffective and incompatible security requirements would only decrease the motivation of the developers towards security and waste the resources of the organization into ineffective and irrelevant tasks.

In general, normative sources (for example, VAHTI 1/2013) seem to contain more recommendations than well analyzed academic papers, or sources that are based on tried and tested software development work (such as Microsoft SDL & Cisco CSDL).

7.1.1 Results of Literature Review

A set of selected sources (articles, books and standards) was collected and analyzed. A table was used to create a cross-reference of sources and their recommended security mechanisms. The security mechanisms that were referenced most frequently are likely to be important for successful software security development. Conversely, the security mechanisms that were referenced only in a few, or just one source, are considered to lack wider acceptance and can be considered optional or experimental.

The results of the analysis should be interpreted with the following limitations:

- The scope of each source varied greatly. Other authors seem to cover process phases that are not strictly within software development process cycle while others remain within the development process boundaries. Possible out-of-scope recommendations were excluded from this analysis (for example, recommendation to define a general strategy for the customer organization, etc).
- The level of detail in each source seemed to vary a lot. Others prescribed controls in great detail (VAHTI 3/2013). Others remain on high level of meta requirements (Siponen, 2002).
- Most sources do not use equal terminology. A manual process of reading, understanding the meaning of the text and manual filtering was performed to understand the purpose of each recommendation. The recommendations were classified accordingly into groups.
- All sources are not peer reviewed publications, i.e. they do not enjoy the level of validation that scientific publications require. Namely

some are publications of individuals or commercial corporations (e.g. Microsoft, 2009), or industry best practice standards that do not contain references to sources.

Despite the apparent difficulties mentioned above, some recommendations clearly seem to enjoy apparent wide acceptance while others do not.

The findings of the analysis are categorized into three categories in the following table. The category names (critical / relevant / optional / unreliable) are used as descriptive labels. Security mechanisms are grouped into categories according to their popularity in the literature. The number of references to the recommended security mechanisms is included inside brackets. The categories in this analysis are:

- **Critical**: mechanisms that almost all authors recommend (mechanisms very likely to be applicable)
- **Relevant**: mechanisms that many authors, but not all, recommend (i.e. likely to be applicable in most cases)
- **Optional**: recommendations that may prove to be useful in a few cases (but not always)
- **Unreliable**: mechanisms that are not likely to work in most cases since majority of authors do not deem them necessary.

Category label	# of ref. sources	Security mechanisms
Critical	11-14	• Threat modeling / formulation of attacker stories (Threat scenarios) (13) • Application risk analysis (11)
Relevant	5-10	• Functional security requirements (7) • Attacker story and user story negotiation or decomposition (7) • Identification of security sensitive assets (6)
		• Establish security design requirements (goals) (8) • Attack surface, boundary protection, firewall requirements (5)
		• Use integrated unit tests to control the quality of the security features (9)

		• Vulnerability and penetration testing, fuzzing (8) • Use of automatic testing tools (7) • Static analysis, code analysis tools (5)
		• Security Coach, include a security engineer in the development / part time (5)
Optional	3-4	• Attacker story – countermeasure checking (4) • Prioritize security requirements (4) • Definition of security architecture (3) • Threat analysis of software components (3) • Review requirements (3)
		• Recovery planning (4) • Use of secure design models or patterns (3) • Defined access levels (3) • Layered security architecure (3) • Strong authentication, avoidance of weak authentication methods (3)
		• Internal review, code review (3) • High level programming languages and tools (3)
		• Security auditing (3) • User acceptance testing / final security review (3)
		• Security awareness training for developers (4) • Information security policy (3) • Provide periodic security training updates (3) • Use of defined libraries (3) • Use of version control and change tracking (4)
Unreliable	1-2	• Too many different recommendations for analysis (93 different)

Table 11: Critical success factors in secure software development (N=16 sources from literature)

Security mechanisms that are labelled under the ”optional” group in the previous table could well be effective in some instances but ineffective in others. For example, conducting recovery planning (4 recommendations) or use of secure design models or patterns (3 recommendations) could be useful in some

cases but we simply cannot draw a conclusion that they could be beneficial everywhere. Conversely, if we wanted to to perform everything possible in terms of security we would be in danger of ending up establishing nothing because of the sheer volume of the work.

In fact, this analysis gives a good reason to doubt the reasons of implementing all available security mechanisms in general. Our risk assessment about Agile principles earlier identified a possibility that too much security can be just as harmful as too little. It could happen that an Agile company loses competitiveness because it creates a too secure code. Agile security expert B commented in his interview that too much security is the reason why an Agile company could lose the advantage in pricing and suffer the loss of customer projects. He stated that: ”there simply is not room in the competitive software development market for excessive security budgets”.

With this Agile Expert B’s recommendation on economic policy in mind, I believe that the 121 recommendations on the table 11 above (19 labelled as Optional and 93 Unreliable) can be reasonably excluded from further analysis. Therefore, we are going to analyze the recommendations from Critical (2) and Relevant (10) categories from this point forward.

7.2 Analysis of Applicable Security Mechanisms

This analysis chapter binds together the earlier work in this research. This is where we propose security mechanisms for solving the perceived problems and try to comprehend the manner in which they portray against software development phases.

We compare the inherent risks of Agile Methods (based on our earlier risk assessment in this research) with applicable and corresponding security mechanisms and create a synthesis where a control mechanisms are proposed for each identified risk (tables 13-18).

Finally, a high-level model is proposed that is based on the previously analyzed data. This is the Agile Security Model that is the main goal of this research.

Agile core principles				
• Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. • Businessmen and developers must work together daily throughout the project.				
Perceived problems	**Impacts**			
Identifying relevant security features and suitable level of security for the customer is insufficient.	If the security features are not identified correctly early in the development process, it may be too late to implement them subsequently. Conversely, too much and too expensive security may be implemented.			
Proposed control mechanisms & benefits	**Dependency**	**Control type**	**Resource**	**Sources**
Identification of security sensitive assets: understand customer's strategy.	Shared resource	Endogenous, strong	Customer	Literature & Agile expert B
Discuss customer risk appetite: understand acceptable price tag of security and how much risk the customer is comfortable with.	Shared resource	Endogenous, strong	Customer	Agile experts B and C
Threat modelling / formulation of attacker stories (threat scenarios) to product backlog: analyze and address high-level threats in backlog.	Common output	Endogenous, strong	Negative security backlog stories	Literature & Agile expert A
Functional security requirements: include necessary security functionalities in backlog.	Common output	Endogenous, strong	Positive security backlog stories	Literature & Agile expert A
Customer acceptance to the implemented security features and residual risks: transfer residual risk to customer.	Sequential	Exogenous, weak	Written risk acceptance	Agile expert B
Notes:	**Software development project phase(s)**			
During the initiation of the software project, the security-related stories must be created and imported to the product backlog. This is a critical for security to become successful. Examples: high-level functional security features (e.g. access control functionalities), non-functional requirements (software must not allow unauthorized access to data) and attacker stories (misuse cases) need to be specified and written.	(X) Initial planning / preliminary analysis (X) Planning & requirements (iterative) ( ) Architecture and design (iterative) ( ) Development & implementation (iterative) ( ) Testing and evaluation (iterative) (X) Deployment			

Table 12: Analysis #1 of perceived problems and proposed control mechanisms for Agile.

<table>
<tr><th colspan="5">Agile core principles</th></tr>
<tr><td colspan="5">• Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
• The best architectures, requirements and designs emerge from self-organizing teams.</td></tr>
<tr><th colspan="2">Perceived problems</th><th colspan="3">Impacts</th></tr>
<tr><td colspan="2">How to design security-related software features in a way that allows flexibility even in the event of radical refactoring? How to establish change control?</td><td colspan="3">Rigid security architecture may be a reason why application cannot be flexibly refactored. This would lead to loss of value to customer. Uncontrolled changes can break the security architecture.</td></tr>
<tr><th>Proposed control mechanisms & benefits</th><th>Dependency</th><th>Control type</th><th>Resource</th><th>Sources</th></tr>
<tr><td>Establish security design requirements (goals): set goals to manage identified customer’s security requirements and threats to create an application security architecture that results in a desired level of security. Revise the goals with the customer.</td><td>Common output</td><td>Exogenous, weak</td><td>Application security architecture</td><td>Literature</td></tr>
<tr><td>Attack surface, boundary protection, firewall requirements: number of possible application, system and network layer threats is minimized.</td><td>Common output</td><td>Exogenous, weak</td><td>Application security architecture</td><td>Literature</td></tr>
<tr><th colspan="2">Notes:</th><th colspan="3">Software development project phase(s)</th></tr>
<tr><td colspan="2">Product owner, architect & security coach should collaborate to do the following:
First, a high-level application security architecture should be envisioned as a collaborative effort (PO, architect & security coach are probably involved).
The creation of architecture should be a product backlog item if it needs to be documented. The architecture should consider all customer security requirements and result in a desired level of security. The components in the architecture should be decomposed to actionable security items in the backlog.
Include practicable steps for minimizing attack surface on application, system and network layers. Import these practicable items to the backlog as actionable items (this is part of architecture decomposition).
Ensure that version control systems are used to control changes in all of the abovementioned system and application security features. Ideally the build scripts (Ant scripts, etc) should include system level change and setup functionalities, and not only the application part.</td><td colspan="3">( ) Initial planning / preliminary analysis
( ) Planning & requirements (iterative)
(X) Architecture and design (iterative)
( ) Development & implementation (iterative)
(X) Testing and evaluation (iterative)
( ) Deployment</td></tr>
</table>

Table 13: Analysis #2 of perceived problems and proposed control mechanisms for Agile.

<table>
<tr><th colspan="5">Agile core principles</th></tr>
<tr><td colspan="5">• Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
• Continuous attention to technical excellence and good design enhances agility.
• Simplicity--the art of maximizing the amount of work not done--is essential.
• Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.</td></tr>
<tr><th colspan="2">Perceived problems</th><th colspan="3">Impacts</th></tr>
<tr><td colspan="2">How to identify, design and implement the right security features at the right time in the development process when they are needed?</td><td colspan="3">Security features and tasks may remain as an amorphous conglomeration of unmanaged stories that is never properly designed, implemented nor finished. This creates technological debt. Also, totally ineffective or incompatible security features may be implemented.</td></tr>
<tr><th>Proposed control mechanisms & benefits</th><th>Dependency</th><th>Control type</th><th>Resource</th><th>Sources</th></tr>
<tr><td>Application risk analysis: the development team collectively analyzes and understands the application architecture and the threats that are relevant to it.</td><td>Common output</td><td>Exogenous, weak</td><td>Application risk assessment</td><td>Agile expert B</td></tr>
<tr><td>Attacker story and user story negotiation or decomposition to (sprint or product) backlog: prioritize security items on the backlog and turn attacker stories into positive actionable security tasks on the backlog.</td><td>Shared resource</td><td>Endogenous, strong</td><td>Product backlog security items</td><td>Literature & Agile expert A</td></tr>
<tr><td>Perform backlog maintenance: make security work ”visible” on the backlog by splitting security tasks into atomic and practical level on the backlog. This enables that reasonable amount of security investment and work will be done.</td><td>Shared resource</td><td>Endogenous, strong</td><td>Product or sprint backlog security items</td><td>Literature & Agile expert B</td></tr>
<tr><th colspan="2">Notes:</th><th colspan="3">Software development project phase(s)</th></tr>
<tr><td colspan="2">Arrange at least one short common session where development team members and the security coach can discuss the application architecture and technically analyze its weaknesses and design viable security mechanisms.
Negative security items on the backlog (i.e. do not let sensitive data to be stored in clear text format) are turned into positive and practical tasks (i.e. implement SHA-512 hashing and salt in storing user passwords)
Backlog maintenance is performed usually by PO and architect. They prioritize items that need to be done, and also decompose security related stories into more practical level and discuss the items with the customer. This makes security work visible to the customer and developers.</td><td colspan="3">( ) Initial planning / preliminary analysis
(X) Planning & requirements (iterative)
( ) Architecture and design (iterative)
( ) Development & implementation (iterative)
( ) Testing and evaluation (iterative)
( ) Deployment</td></tr>
</table>

Table 14: Analysis #3 of perceived problems and proposed control mechanisms for Agile

Agile core principles				
• Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. • The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. • At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.				
Perceived problems	**Impacts**			
How to manage the human aspects of secure Agile development, the aspect of security awareness?	Lack of security awareness may prevent any security activity from taking place. At minimum, it will slow down the progress.			
Proposed control mechanisms & benefits	**Dependency**	**Control type**	**Resource**	**Sources**
Security Coach, include a security engineer in the development / part time: provide specialist advice for the team.	Shared resource	Exogenous, weak	Security coach	Literature & Agile expert C
Establish supportive, collective work environment that enhances security learning: enable learning security skills on organizational level; build secure development capability.	Shared resource	Endogenous, strong	Supportive environment	Literature & Agile expert B
Empower developers to act on security issues: improves developer efficiency in solving security issues.	Common output	Endogenous, strong	Secure code	Agile expert A
Gain understanding of threats and security architecture: the developers share an identical vision of what and how software should be protected.	Common output	Endogenous, strong	Secure code	Literature, Agile expert A
Notes:	**Software development project phase(s)**			
The project owner should appoint a person for security in the development team, preferably someone with personal ambition towards software security. The PO should ensure that the working environment supports security learning. Developers should have a permission to acquire literature, study and use their working time for learning how to create secure code. This is an issue of providing necessary resources. Security courses and training can be used to supplement the learning. Rotating roles of the developers may also be helpful. The PO should also empower developers to solve low-level security issues independently (at least on code level). The developers should gain a shared, collaborative understanding of the application security threats and the manner in which the security architecture should work. This enables them to work as a team towards common security goals.	( ) Initial planning / preliminary analysis ( ) Planning & requirements (iterative) (X) Architecture and design (iterative) (X) Development & implementation (iterative) ( ) Testing and evaluation (iterative) ( ) Deployment			

Table 15: Analysis #4 of perceived problems and proposed control mechanisms for Agile.

Agile core principles				
• Working software is the primary measure of progress.				
Perceived problems	**Impacts**			
How to ensure that software is not infested with security vulnerabilities after the point where software is deemed to work?	If software vulnerabilities are not managed during the development, they are bound to emerge in the software product.			
Proposed control mechanisms & benefits	**Dependency**	**Control type**	**Resource**	**Sources**
Use integrated unit tests to control the quality of the security features: enables change control for monitoring that security features do not break when something changes in the code and ensures that they work as expected.	Sequential	Endogenous, strong	Unit test automation	Literature & Agile expert A
Vulnerability and penetration testing, fuzzing: provides feedback about unexpected and unwanted software behaviors. Vulnerabilities in software behavior and business logic are identified and can be corrected.	Shared input	Exogenous, weak	Testing reports	Literature & Agile expert A
Use of automatic testing tools: can include regression testing, performance testing & c. Their use ensures that the general quality of the software remains on an adequate level.	Sequential	Endogenous, strong	Test automation	Literature & Agile expert A
Static analysis, code analysis tools: pinpoint coding errors that can be inspected and corrected by the developers.	Shared input	Exogenous, weak	Testing reports	Agile expert A
Notes:	**Software development project phase(s)**			
After security features are implemented, they should be monitored by using automated testing technologies; unit tests, regression tests, performance tests & c. are few recommendable techniques. Automation is the key to high performance because it can give early signals about the manner in which the development team collaborates and ensures that the features do not break during the development. Exploratory (investigative) testing cannot usually be completelly automated. When such testing is required, vulnerability testing, penetration testing, static code analysis and fuzzing tools can be used to create an additional feedback mechanism into the development process. Note! These techniques can be automated at least partially, for example, fuzz testing can be automated and resulting error conditions investigated. Also, any signature-based testing methodology (that uses a black list approach like vulnerability scanning) can be automated.	( ) Initial planning / preliminary analysis ( ) Planning & requirements (iterative) ( ) Architecture and design (iterative) ( ) Development & implementation (iterative) (X) Testing and evaluation (iterative) ( ) Deployment			

Table 16: Analysis #5 of perceived problems and proposed control mechanisms for Agile.

7.3 Detailed Description of Applicable Security Mechanisms

This chapter explains in detail the phases of software development process and the security mechanisms that are critical for creating secure software using Agile Methods. The concept of ”process phase” is used here to describe the types of necessary activities during the software project. The phases may overlap, or their order may be different and they may iterate several times.

7.3.1 Initial Planning Phase

Identification of security sensitive assets

According to the Agile Expert B the main goal in the beginning of the software project is to understand the customer’s strategy. By strategy he meant that we should understand the business goals of the customer in the whole software project. He explained that ”…if you are trying to produce a new kind of a dog food recipe, then you better first find a recipe that the dogs like and the pet owners are likely to buy. Unless that criterion is met, it is irrelevant if the dog food happens to include a disliked flavor of cross site scripting (XSS)…”

Thusly the customer is the only source who is able to specify the business context of a specific dog food recipe. The product owner should discuss with the customer the kind of data the application will store, process and transmit and if any of that data is critical from business perspective (dog food recipe). The goal of this identification process is to understand whether some part of that data is sensitive, or requires more security than the rest. Eventually, this understanding of high-level requirements should result in creating high-level security–related product backlog (figure 7). According to the Agile Expert B, this is critical for defining the desired level of security and the acceptable price tag for the security expenditure. Without understanding what is essential for the customer security, investment cannot be focused on important parts of the application.

Agile Expert C commented that in one case a preliminary study needed to be conducted to identify legal requirements for transactions and responsibilities with regards to payments. According to C, in the initial planning phase it is usual to "…consider what systems are involved and what kind of information is stored in them, and what protections should be in place to protect the information in a client-server architecture. …We have identified what pieces of information are more critical than others, and what are the most sensitive pieces of information in the systems." He described the process of asset identification: "The customer has described their own view about potential problems, and which assets they consider to be critical. They (customers) usually understand better than external consultants how the system is intended to be used. The usual problem is that the customers don't always understand the assets on correct level: They might start with a technical network diagram but they really should first understand what is the business context and the value to the business instead of technical approach." In his opinion the reason for the usual technical approach was that people consider information security to be primarily a technological or IT issue, and people who implement these systems are usually engineers. However, the owner of the assets and their problems is an entirely different stakeholder from the engineers, it is the business. This difference in roles was manifested when C told a story about a technical engineer / architect who later became the product owner. According to him, the same person used to consider that all technical security solutions should be developed equally when he was in a technical position. As soon as the person started working as a product manager he suddenly realized that he couldn't afford to buy everything. He started to consider business benefits of each security item that are correlated with security goals.

The point is that without understanding the type of information in the system it would be pretty much an ad hoc choice to implement any security. For instance, in one case we found that an organization was planning to transmit confidential data across public internet to mobile handset clients. Their initial plan was to allow access to the resource by plain username and password authentication without any further protection. This would have allowed them to

access some very sensitive resources. Discussions eventually led to realization that the mobile device and the client application were not the only assets to consider, but that they should also assess the value the sensitive resource to which they were allowing access. Thusly the problem space was extended from the mobile device and the application to cover the resource and its value to the customer. This change of view in the asset value completely changed their mindset towards the level of required security. The end result was that they decided to implement rather high level of security for the application architecture with TLS encryption and mutual certificate-based authentication scheme.

Discuss customer risk appetite

According to the Agile Expert B, the customers have various degrees of risk appetite. By definition, risk appetite is the level of risk that an organization is prepared to accept, before action is deemed necessary to reduce it. Understanding risk appetite is essential to understanding what is the adequate investment in security, compared with the amount of risk the customer is comfortable. Dhalla's (2009) analysis of reputational risk describes the relation of product quality to reputational risk: "it is likely that failure of … a new product offering will send a signal of low quality, creating reputational risk." In other words having insecure products can be interpreted as a signal of low quality.

Agile Expert B said that his customers are roughly divided into three kinds of attitudes towards risk:

1. Willing to invest in software security (wanting to avoid risk)
2. Cautiously willing to invest in security, often not wanting to pay for extra security (cautious preference to take some risk)
3. Innovative, seeking high business rewards and reluctant to invest in security (accepts greater inherent risk)

Also according to Agile Expert C, Agile development companies and customers who buy their products come with various degrees of risk appetite. He recalled several projects to a certain large customer where he was involved in performing security audits to software just prior to deployment to production. In every case when this customer was involved he had found several serious security issues from the software and recommended that "… this is not the recommended way to create secure software – they should start with security requirements much earlier in the development". He considered this to reflect the rather high risk appetite of this customer because the customer "…didn't consider the security requirements necessary because developing them would be an additional cost factor. For the customer, it was cheaper to audit the products prior to production."

The worst case scenario was, according to Expert C, when "… an Agile development company delivered a solution to a customer. We found a terrible amount of horrible security issues in the audit prior to deployment. The developers soon told that they had fixed the issues. We re-audited the software and found some of the old issues and a handful of new ones that they had generated when fixing the old ones. This iteration of audit-and-fix continued until serious issues were fixed." C also stated that those developers admitted later that "…they knew that these issues were present in the code, and that they were aware of the findings of the audit prior to auditing. Despite having this knowledge they decided to deliver the software to the customer and did not actively do anything to address the security vulnerabilities." Finally, they admitted the existence of the the known problems to the customer only after the audit had taken place and identified the problems.

Establish security design requirements (goals)

Establishing security design requirements, or goals, is necessary for constructing an application security architecture that results in a desired level of security. According to Sindre & Opdahl (2002), security goals should be defined for

each identified asset (such as user database), and threats can be understood as obstacles that prevent reaching those goals. Mead, Hough & Stehney (2005) stated the goals should be identified and negotiated with different customer stakeholders since otherwise it is not possible to know the relevance and priority of any subsequent security requirements that are generated afterwards.

According to Mead (2010) security goals are determined first, before requirements, so that the team will understand what outcome the security requirements are to support." Goals of the security design are closely related and necessary in formulation of attacker stories. Namely we may have an asset with a frontend server and a backend user database that should be protected against threats. Our design goals could be expressed as follows:

- "Prevent unauthorized access to sensitive user data stored and processed by the application"

This meta requirement could be further broken down into security design goals such as:

1. Design the high-level security architecture only to allow direct access to the frontend application while preventing any unauthorized direct access to the backend user database.
2. The frontend application should provide session control, user authentication and data input controls. The underlying application server should be hardened against attacks on system level.

These goals of the security design could be further broken down into smaller parts and included in product or sprint backlog.

According to Agile expert C, an example of a security goal in one customer case was that the system should be able to prevent fraudulent transactions from taking place. He told that "… this goal was not obtainable just by means of technical controls. Fraud control would have to be addressed also on design level when roles in the system are designed, for example by means of segregation of duties". C also commented that often organizations seem to have

rather general goals like “software must not allow unauthorized access to resources”, or “the services must be available 100% of the time”.

Expert C’s experience was that usable methods for defining security goals are interviews of customers, or the use of formal checklists. His opinion was that direct inquiry about security goals would not perhaps work because few customers understand what that term means in practice.

The importance of security goals is to explain why all security features of the software should be implemented.

7.3.2 Planning & Requirements Phase

Functional Security Requirements

Functional security requirements are formulated early in the software development. The requirements should be included in the product backlog, either as separate stories, or as part of other use stories. According to Peeters (2005), it is common to find security-related details inside user stories that describe in high level what the user does. Inclusion of the security functionalities in user stories enables integration of security in software development and possibly prevents security from becoming externalized. Functional security requirements are positive, i.e. they describe what kind of features should be implemented in the software.
The following figure displays an example of functional security requirement that can be written for a login functionality of an application.

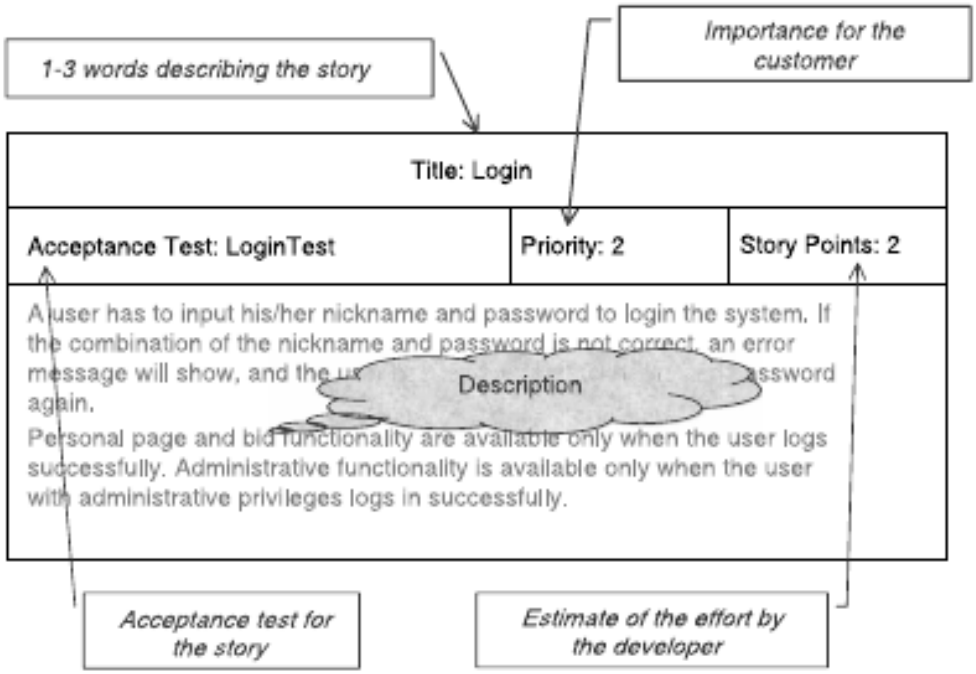


Figure 11: Structure of a user story (De Lucia & al. 2008)

According to Agile Expert C, many customers who acquire software from software development companies use some kind of formal requirements as attachments in their agreements. Often those requirements are rather technical, including protocol specifications, message sequence charts, or input control requirements. A good requirement is not always enough though. In some instances, C told that he even encountered contractor employees who "…did not know the basics of HTTP protocol although they were tasked with developing a secure web application. They did not know the basics of HTTP GET, POST or cookies." He ended up arranging a short training session about protocol basics to those contractors. Then he asked the contractors to present their current code and found out that "…it was as it is usually always (horrible). There was no sign of centralized control mechanisms in the code. I was able to help them to learn how to implement the security requirements properly."

The level of detail in which functionalities are described varies greatly. High level features can be as abstract as implementing a full-blown PKI architec-

ture for a large application. In detailed cases, it has sometimes been necessary to describe in what manner the application should implement one phase of cryptographic operation in one step of its message sequence flow. This could be described in a message sequence chart like described above. It seems to be important that the stories are written in a manner that enables the product owner and the developers to understand them. Usually it is also necessary to explain the requirements directly to developers because otherwise they may have difficulty understanding them.

Threat modelling / formulation of attacker stories

Threat modelling is a top-down activity that results in formulation of high-level attacker stories. Setting security goals and performing detailed risk analysis on application–level are related to threat modelling. Threat modelling should result in security-related items on the product backlog. According to Binder (2004), lack of perception about security problems can lead to inadequate analysis of the problems and communication failures. This could escalate further in application, subsystem and class–level problems.

According to Peeters (2005) attacker stories ”identify how attackers may abuse the system and jeopardize the stakeholder’s assets”. Peeters also proposes that attacker stories should be brief, and they should be ranked and scored similarly to user stories (use cases). Peeters also proposes that each attacker story should assess the likelihood and impacts of successful abuse. Furthermore, the scoring should be commensurate with the user stories. Agile Expert A commented that ”… threat analysis is the most important phase of secure software development…”.

Writing attacker stories does not differ considerably from writing user stories. We can use exactly the same model that is used to describe user stories. This enables us to create both functional and non-functional security requirements with similar scoring, priority structure, testability and also to able to com-

municate them to the developers in an easier manner. According to Sindre & Opdahl (2002), visual diagrams can be used to visualize the relations between user stories and attacker stories.

It should be noted that attacker stories are usually negative. This means that they describe something that should not happen, and cannot be implemented as such. Agile Expert A commented that "… if the (security) feature is described so that something should not happen you cannot simply implement it because we implement positive features. The code is always doing something."

The following is an example of formulating a non-functional security requirement:

Title: Do not allow input attacks against the application		
Acceptance criteria: Implemented input validation controls	**Priority**: high	**Story points**: __
Attack origin: internet (WAN) **Attack vector**: exploitation of application input vulnerabilities. An attacker may craft input strings in order that the application constructs unwanted commands. For instance in SQL injection attack statements are constructed based on the input, and the resulting SQL statement performs actions other than those the application intended. (CAPEC-1000) **Impact**: Very high. Input attacks are common and basic to exploit. (i.e. simply inputting 'or 1=1-- string on input field can identify a SQLi vulnerable application). Therefore, this should be given high priority.		

Table 17: Example of a non-functional security requirement (CAPEC-1000)

The following diagram displays the relations of positive and negative security features and their relation to software development activities.

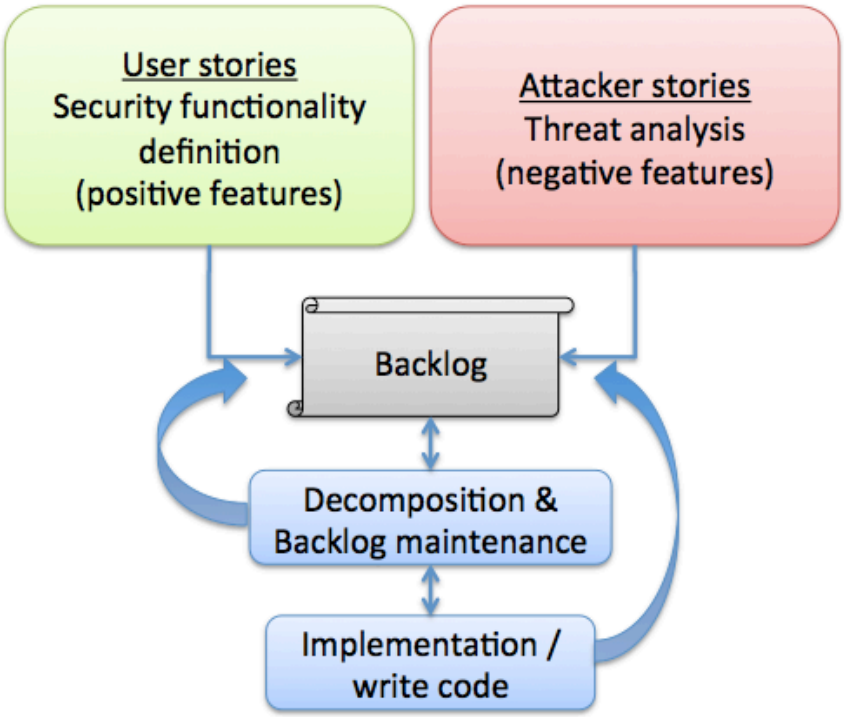


Figure 12: Relation of positive and negative security features.

Agile Expert C considered the process of threat modeling / attacker story formulation as a process that identifies points where something could go wrong. He told that he has experience from software development projects where they "analyzed various threat scenarious before implementing anything". He also made a remark that at this time the price of security is usually not yet a concern. "Anything that could be perceived as a threat should be analyzed. After that it is possible to do prioritization and think about which issues are most critical."

In C's opinion the usual case with software projects is that they normally don't do much in the way of threat assessment or formulating attacker stories: "…and when they do, it is usually quite minimal. They can simply decide that using HTTPS is enough, or in best cases they make a reference to OWASP top 10 vulnerabilities: Those are the threats that we should address." He stated similarly to Expert A that the problem with negative requirements is that they describe something that software should not do. According to him,

this is sometimes very difficult to invert into concrete, positive functionalities.

Expert C also had good experience from doing threat modeling by identifying roles that are involved in the use of the software. He told that an analysis of unintentional or intended misuse cases was usually conducted and a model was developed based on those interactions. This approach is consistent with Pauli's & Xu's (2004) misuse case modeling approch. An example of misuse case threat modeling is displayed in the figure below. The white bubbles describe software functionalities and the black ones intentional or unintentional possible misuse cases.

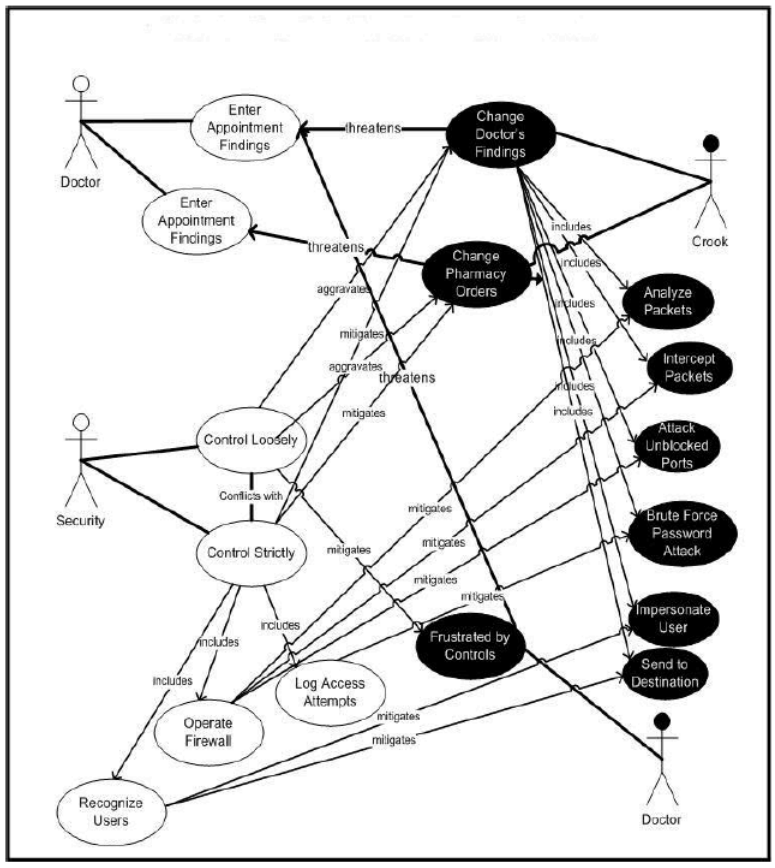


Figure 13: Misuse based threat modeling (Pauli & Xu, 2004)

Application risk analysis

Some form of threat or risk assessment was recommended by almost every author in the literature. The same recommendation was also given by all interviewees. It appears that there is a wide consensus that attacker stories (analogous to abuser stories or misuse cases) should be formulated based on the perceived application threats (or risks). The most important benefit from the application risk analysis is that the development team collectively analyzes and understands the application architecture and the threats that are relevant to it. Agile Expert A commented about the process of application risk analysis thus: ”… commonly, there is this one guy (with the development team), a whiteboard, and we draw a diagram, which is the way that most companies do it in the world, for example this is how Microsoft does. We draw data flow diagrams that display the interactions between the components of the system, databases, systems and software components. In very complex cases, with complex protocols, Message Sequence Charts can be used. Microsoft does this with STRIDE.” He also had an opinion that it is basically irrelevant which application threat modelling method is used as long as it serves the goal of increasing comprehension of the threats, and enables communication about the threats. According to Expert A, the most important benefit from application risk analysis is that ”…the developers learn the mindset of the attacker by using a structured method. This way the developers can intuitively avoid design-level security problems in their work even before those problems end up in these application-level risk analysis diagrams.”

The Microsoft STRIDE is a noteworthy method for performing the application risk assessment (Hernan & al. 2006).

According to Agile Expert C, application risk analysis is best to perform in close co-opration with the developers. He told that it has been successfully performed even as an external consultant -led effort, but considered it better to involve the developers in the process because they would need to understand the risks anyway. He commented: “…usually the developers have

learned to think about application risks. The increased understanding manifests itself when they learn to ask the correct questions from us."
Regretfully, Expert C had found that most developers don't do any kind of application risk analysis. In his opinion the problem was that "…people think that they are not required to do risk analysis because no-one else seems to be doing it either. And consequentially, no-one usually requires that it would be done." His opinion was that the root of the problem was that "…very few developers have taken the time to learn about application risk analysis. And there are very little requirements that developers should learn to do this. Consequentially, they often skip application risk analysis entirely."

Learning to think like the adversaries is the key to successful application risk analysis. It is also a key to a successful secure design because unidentified application-level risks can't be mitigated effectively.

Attacker story and user story negotiation or decomposition to backlog and performing backlog maintenance

The benefits of decomposition and backlog maintenance are to make security work "visible" on the backlog by splitting security tasks into atomic and practical level on the backlog.

Attacker and user story negotiation and decomposition are processes where:

1. negative security stories can be integrated into other user stories or
2. decomposed into smaller parts and turned into doable positive backlog items

The process of decomposition is usually a dialogue between the product owner, a software architect and the development team. The backlog maintenance should be performed regularly, and time should be allocated for executing it. According to Agile Expert A the process of decomposition leads to removal of attacker stories from the backlog and replacing them with doable (positive) development tasks (Figure 13).

The following table uses our earlier example of an attacker story. In this case, the attacker story has been decomposed and inverted from negative into a positive development task that could be implemented in the next sprint.

Title: Do not allow input attacks against the application		
Acceptance criteria: Implemented input validation controls	**Priority**: high	**Effort**: 12 days
Attack origin: internet (WAN) **Attack vector**: exploitation of application input vulnerabilities **Tasks**: Implement input validation of user-controlled data before including it in a SQL query. Use parameterized SQL queries in the application logic (e.g. PreparedStatement in Java, and Command.Parameters.Add() to set query parameters in .NET).		

Table 18: Example of decomposing attacker stories to backlog.

Agile Expert C also had the same view with Expert A that inversion of negative security stories was crucial. Expert C told: "…you first need to understand the root cause of the problem. For instance, what is the thing that causes the injection to occur? Only after understanding what is happening you can remediate the issue. For instance, you may come to realize that injection is only possible when it is possible to bypass data type validation in the application. As a result of this understanding the injection becomes really a problem with data type validation." Again, he told that inversion of attacker stories usually necessitated specialist knowledge: An average developer may be unable to understand how to invert these requirements. What is required is that a person does the inversion who has experience from doing it. C explained that the most common error is to start creating a blacklist of things that should be denied in the software: "as long as you describe things in negative terms it will be difficult and ineffective. A whitelist approach is much more effective." In his opinion, the negative requirements cause the developers to learn a wrong mentality of denying, or blacklisting unwanted things instead of developing more effective whitelisting functions.

Expert C told a story from a case where another company had audited a web application and found a cross site scripting (XSS) vulnerability. The developers had created a patch that filtered some unwanted characters and strings from the input that prevented that specific attack. The problem in that instance was that they did not entirely undestand the real problem and had the mentality of blacklisting unwanted behaviors. When the application was audited by C, he simply double encoded the same input attack messages and bypassed the blacklist protection. He explained that the problem with blacklist approach was that it is unable to address any future problems because only known problems can be listed. A good mitigation to this XSS vulnerability would have involved inverting the protection requirement around: They should have defined what kind of input is allowed to the application and allow only that sort of input.

Expert C made an example of inverted positive input validation control: “If we expect to receive only text input to the application we should define the application only to allow letters in the input. The other, flawed blacklisting approch is to make a catalogue of all the letters that are denied. This is an insane idea which is implemented only by those who can’t properly invert attacker stories to positive items.”

7.3.3 Architecture & Design Phase

Attack surface, boundary protection, firewall requirements & gaining understanding of threats and security architecture

The result of successful application security architecture is that the number of possible application, system and network layer attack vectors is minimized. The intention is to reduce the overall complexity of the whole architecture and make it better protected against attacks. The principle of attack surface minimization should be considered from a holistic perspective in the architecture (examples):

- **Application level**: minimum number of application functionality is exposed to users. For example, any unnecessary testing & debug functions are removed before deployment. Security validation checks are performed against all remaining user actions.
- **System level**: all system services, modules and libraries that are not necessary for the intended application functionality should be removed or disabled. Systems should be configured (hardened) against attacks.
- **Network level**: network access to the systems and applications should be limited to minimum by network–level filtering, namely access to administrative application functionality should not be openly exposed to all users of the application.

Agile Expert C explained that "a (security) architecture is just a figment of imagination that can be implemented in endless different ways". Therefore we should understand that security architecture is a flexible concept that needs to adjust accordingly to the goals of the customer. According to expert C different kinds of modeling tools have been used to create security architectures: Message sequence charts (MSCs), logical diagrams with systems and their interactions and high-level API descriptions. He explained that architecture usually never goes to level of details. It explains what each software module does and is responsible for security-wise. He also divides application security controls into main categories:

1. Controls depending on the case. For example, role-based access controls, fraud control, or a support for legacy protocols. These controls have to be designed always individually.
2. Generally applicable controls, especially in the world of web applications. For example, input validation and session management controls. For the general controls, he suggested that security design patterns should be used whenever possible.

While the word architecture may sound like a written document, it is also quite possible that the architecture is a virtual construct of backlog items and exists only in the minds of the developers. In our experience the level of re-

quired formalism depends on the case. In most cases the organizations have not deemed it necessary to create dedicated written application security architecture. Our view is that the application security architecture is primarily a means to increase developer's awareness on the important matters in security and the reasons behind it. Similarly, according to Pauli & Xu (2004), the architecture enables the developers to understand the security architecture and clears up confusion among the developers. They also propose that inclusion of security in the software architecture makes applications more resistant to vulnerabilities.

Agile Expert B explained that it can be wise to ask for formal customer acceptance to the security design (architecture) in the design phase of the project: "…with the most security aware customers, one should have written acceptance from the customer about the architecture design." If the security design contains any known weaknesses, the customer should be aware of them. For instance, if unencrypted FTP integrations are to be used over public networks, then the security implications should be accepted by the customer.

Figure 14 displays the relations of security architecture in software development. Threats and risks are usually initially unknown until they are analyzed. There is always some amount of residual risk that needs to be accepted because of economic constraints. After performing an initial architectural analysis, the most important security features can be implemented. As the customer's goals can change during the development process, some security features may need to be redesigned and refactored because the business objectives may change during the developent. Parts of the security architecture can be considered to be "Done" when the residual risks have been mitigated to an acceptable level by implementing a necessary amount of security features.

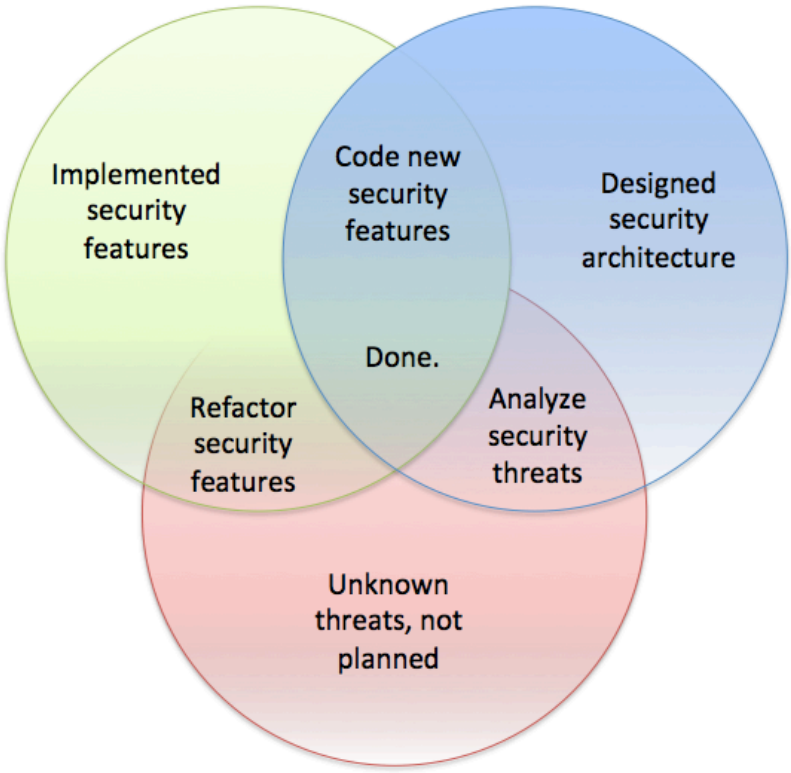


Figure 14: A dynamic view of application security architecture.

7.3.4 Development & Implementation Phase

Security Coach, Include a security engineer in the development / part time

BITS (2012) proposes that ”Application security champions / Mavens” should be nominated. Additionally, they propose that ”Advanced champions” and ”Technical security officers” should play a role in software security too. The recommendation to have a dedicated or a part-time security specialist in the development team was also given by the Agile Security Experts A & C. Agile Expert C explained that he has acted as a security coach and done lecturing, threat modeling, trained the developers to use security testing tools and performed code reviews to code that concerns critical components of the software.

When quesioned about why companies use security coaching Agile expert C explained: “…companies who develop software products are nowadays quite interested in security coaching because they have compliance requirements for security. And their customers want to use software that is secure.”

Expert C explained that therei is usually lack of security skills: “It is common that customer does not have anyone with real security capability. It is quite common that developers imagine to be knowledgeable about security matters. But when we inspect things and dig deeper, we notice that they really don’t have a deep understanding about the issues. This may cause a false sense of security because internally developers would trust this person in security matters because this person has a kind of a security monopoly in the team. They don’t question him because no-one in the team knows security any better.” Next, he gave an example of such a case: “… in one instance I was involved in developing a security protocol with this one security development guy. I wondered why he had ended up encrypting data traffic only in one direction but not the other way. He did not understand how cryptography works. We demonstrated that encryption can be done bi-directionally.” Thus, our experience from the field shows that most developers do not have the necessary specialist knowledge about the matters of technical security for securing the systems and applications they create. It is not uncommon to find false sense of security among the teams. They often truly believe that their software and systems are secure while the exact opposite is the truth.

Sometimes developers are aware of their lack of security knowledge. In one particular instance, the Agile Expert C had been coaching an Agile customer and remembered that the developers ”said that they understood that they do not understand all the security issues that they should”. The developers also had told him that they realized all too well that ”…fixing security issues after deploying code was a problem that needed to be addressed”. Expert C said that the dialog with the developers (coaching) led to identification of several problems in their software and development practices. Next, he simply created an inventory of identified problems and presented it to the product owner.

Soon, those problems were included into their product and sprint backlogs, and they started fixing the problems.

The role of a security coach is both technical and social. According to Agile expert C, several experienced Agile developers have expressed opinions that "…software projects should include a person who understands information security. This person should not just do security testing but also freely communicate with people in the team."

Establish supportive, collective work environment that supports security learning & Empower developers to act on security issues

Developers face today a vast diversity of various platforms that should be supported by their software. For example, a java application should often be portable to several mobile platforms and desktop systems. This puts the developers in a difficult situation: how should they be able to secure all of those diverse environments where their software is being deployed? The BITS framework (2012) proposes that: "integrating IT risk controls within the software development process as a part of a software security program requires ongoing education for key stakeholders…" Agile Expert B also explained that his organization addresses this difficulty by enabling organizational learning on all possible levels. They build a supportive working environment where developers are allowed to use their working time for studying security whenever it was deemed necessary. He mentioned a few useful practices that have proven helpful: they maintain a library of various books and allow purchases of new ones at work. They also rotate working roles so that, for example, a skilled software developer can take a role of an operating system-level designer in the following projects. This allows the members of the development teams to learn from each other and improves their ability to work together. According to Agile Expert A, this practice of "collective working" also benefits the teams because they become highly skilled, and therefore the project owners can entrust them with more responsibility with regard to security decisions (on code level, for example). Also Vähä-

Sipilä (2011) recommends that a practice known as ”empowerment” in Agile is beneficial for security as long as the limits of the responsibilities are clearly defined.

One motivational factor was observed in a case where Expert C met developers who had been given the responsibility for securing their product. None of the developers considered themselves security professionals. They wanted to seek verification and feedback for their designs from C. According to C, they were smart people who were unsure about their security capabilities, and had a very positive attitude towards security. This was positive because they were quite receptive towards security proposals.

Often organizations have failed to implement security because formally, nobody has been in charge. Security responsibilities are often informal and lack the necessary management support. Agile expert C’s observation was that “… often the problem is that there sometimes is a capable internal individual in charge of software security. Sooner or latter it happens that this person changes duties or leaves the company. Thus security is left to nobody because his position usually wasn’t formal. His absence may not cause any pressure towards recuiting a replacement.

7.3.5 Testing & Evaluation Phase of the Project

Use integrated unit tests to control the quality of the security features & Use of automatic testing tools

Agile relies heavily on the idea of minimizing the work of people and automating tests as far as technically possible (Agile Manifesto principles). Test automation ensures that whenever a software feature is implemented, a number of applicable automated test cases are created. Test case automation should ensure that whenever the software is changed nothing breaks. If the automated tests pass and the build doesn’t fail, the developers can accept the chang-

es. Some Agile methodologies such as TDD (Test Driven Development) go as far as proposing that test cases should be created prior to software features. Test automation also has several benefits security-wise. Security features are not different from other software features in that their changes need to be controlled in test automation. For example, new software features may have impact on the security architecture and functionality in the software. The software changes may cause functional problems in security features, circumvent security controls, or create new software vulnerabilities.

We have observed that customers have variations in how diligently they do testing and how they set up their testing environments. According to Expert C, in some instances the companies have created several different layers of testing environments for each of their products. In a different instance, the only real testing environment was the customer's development environment. In that case, what they called "test systems" were in fact replicas of production systems that did not have contain any test automation functionality.

Our experience about test automation is that it is a basic prerequisite for any effective automated security testing. If the company does not have any kind of a mature test automation environment, also automation of security tests is bound to fail because the organization simply does not have tools and test environments for doing the automated security testing.

Vulnerability testing, penetration testing, fuzzing and static code analysis

The topic of vulnerability testing, penetration testing and fuzzing contains a wide range of good practices. In this context they shall simply be called 'security tests'. Whereas test case automation is primarily used to control changes in software, security tests are aimed at finding unwanted and unexpected behaviors. A brief explanation of each type of security testing activity is in order:

1. **Vulnerability testing**: is focused on finding known software and system vulnerabilities. Tests are usually run by using automated scanners that can run tens of thousands of test cases against the systems and applications. Vulnerability testing is limited to finding only previously known software weaknesses. These types of tests can be semi-automated or fully automated depending on the case.
2. **Penetration testing**: is performed by highly skilled testing personnel who use their experience and intellect to detetct flaws in software and systems. Whereas other test types are unable to find flaws in business logic or other errors that are difficult to test automatically, penetration test may expose these weaknesses. Penetration tests are also able to verify and demonstrate the impacts of software security vulnerabilities. Other test types lack this step of demonstration and impact verification.
3. **Fuzzing:** is a test type where robustness of the application is stressed by sending ”…loads of crap against the interface – preferably crap generated by a good fuzzing tool” (Vähä-Sipilä, 2011). The benefit of doing fuzz testing is that it is the most efficient method for identifying unknown software vulnerabilities and enables catching regression errors early (Vähä-Sipilä, 2011 & SANS SEC660, 2013). Fuzzing can be automated, semi-automated or performed manually.
4. **Static code analysis:** is the analysis of computer software that is performed without executing programs using automated tools. The analysis can be performed either against uncompiled code, or by disassembling the compiled code. The analysis reveals programming errors that are otherwise difficult to spot, for example, buffer overflow conditions (Wikipedia).

There is an underlying fundamental difference between security testing and unit testing. Unit tests typically cover a limited number of test cases and software functionalities. They are deterministic in nature, in that they focus on testing the known functionality of the software. Security testing, on the other hand, focuses on functionality which ”should not be there”, and this is

where the emergent nature of software security appears again: as opposed to deterministic testing, security testing is stochastic in nature. The problem space of security testing contains practically infinite number of test cases that can never be fully explored. According to Siddhu (1989), there are SA^{ES} possible implementations for a machine that has S states, E events, and A actions. The problem space increases very quickly with the complexity of the machine, for example: 5 states, 2 events and 2 actions there are over 10 billion different implementation possibilities. Therefore security testing is inherently different from unit testing. Only a small subset of all possible implementation possibilities can be realistically covered by the security test cases. Thusly automatic security testing approach should be desired over manual testing: the modern applications contain so much complexity that all possible testing paths can never be perfectly covered. Automatic and intelligent sampling and test case generation is required. This also explains why fuzz testing is so effective in finding vulnerabilities where humans have failed to spot them: machine-generated ”fuzz”, or ”crap” is able to explore much wider amount of testing paths than mere manual intervention.

Agile Expert C had experience from being involved in developing automated security tests for several Agile companies. In one instance the customer had a mature unit- and systems testing environment. After performing a technical penetration test C discovered software data input validation vulnerabilitites that they did not have test cases for. After informing the customer about the vulnerabilities they soon developed new test cases for those vulnerabilities, fixed the problems in the code and the new test cases passed after that. In our experience this is the usual approach to security testing: after finding a problem, Agile developers retroactively develop a few test cases until the tests pass. Unit testing is the usual way for implementing the automated tests.

In another instance, Expert C’s customer who considers security in high regard was performing automated security tests with several different methodologies: “They did unit testing of web application functionality via a tool that au-

tomated browser functions. Misuse cases were automated by using python scripting. And they used some internally developed fuzzers to test the robustness of their code. But they didn't use automated security scanners in their testing." This case is indicative of proactive approach to software security testing. They did not rely on just fixing the problems but used several different approaches to proactively test for security flaws. They could have done even better by using automated security scanners in their security test automation.

Our experience from the fields of vulnerability testing, penetration testing, fuzzing and static code analysis is that most companies are usually unable to perform these activities on their own. Usually they decide to obtain external specialist services instead of performing these actions themselves. Expert C summarized his experiences from several instances: "According to my observations, most organizations don't do security testing at the time of software development. They do it when development has been done. It is too late to do the testing when the development is finished as in waterfall model."

In some instances, we have been involved in all stages of the software development life cycle with Silverskin's customers. In those instances our security testing at the time of deployment has shown that software is designed and implemented to be much more secure and resilient than in average. (Silverskin, 2012)

7.3.6 Delivery Phase

At the delivery phase of the Agile software project, it is important that the customer comprehends the risks of the current implementation prior to deployment. Therefore a formal acceptance should occur. It is noteworthy that this acceptance is closely related to formal customer acceptance of the application security architecture. Ideally there should not be discrepancies between the accepted security architecture and its implementation at the time

of delivery. However, the customer may have already made risk decisions at the time of architecture design that cannot be easily changed at the time of delivery. Agile Expert B explained that if customers ”…choose to amputate their own leg they will be angry at the seller of the chainsaw.”

According to Asthana & al. (2012) Agile uses sprint cycles to track progress, and consequently it is necessary to use the sprint backlog for tracking unimplemented security issues at the end of each sprint. Undone security work should go back to the backlog. This ensures that related risks can be tracked and communicated to the customer. As each large product release usually consists of various sprints, a release may include multiple known risks and issues. Any risk that may have impact on business objectives of the product owner or the customer must be approved prior to release. Asthana & al. also propose that if remaining residual risks cannot be approved by the PO or the customer, the code cannot be released. The only viable option in this case would be to add security related tasks to the backlog to mitigate those risks.

Another benefit from well working sprint / product backlog management is that all security related features, risks and issues are documented in the backlog. This enables creation of security related documentation during the development instead of creating the documentation externally outside of development. As opposed to waterfall model, the security documentation in Agile is usually integrated into the backlog items instead of separate documents. Hence a security compliance assessment can be performed by comparison of the application security architecture against the implemented security items on the backlog. If the customer desires, the product owner could create a backlog item that requires that such internal review is performed and its results are documented and handed over to the customer. This would provide proof that the customer security requirements are included in the security architecture and implemented in the software.

Sometimes security risks are not acceptable by the paying customer. Agile expert C was involved in a project that failed: “The customer had audited their

software at the time of delivery and discovered a load of vulnerabilities. This happened a week prior to go live -date. In fact, several projects from the same organization always seem to follow the same formula. Testing too late has always led to delays in go live -dates. The absolutely worst case was when the customer had to re-audit the software so many times that they ran out of budget and had to trash the software project. They never went live with that one." This worst-case instance is an example from a case where the residual risk was not clearly acceptable by the customer. They rather trashed the project than accepted bad level of security in their product.

8 The Agile Security Model

This chapter introduces the holistic Agile Security Model (figure 15):

1. The structure of the model is based on generic phases of Agile software development process. The development cycle describes the iterative nature of Agile development (iteration in the figure 15).
2. Applicable control mechanisms and their benefits are attached to each phase for each process. Note: since Agile development often proceeds in incremental iterations, and its phases are not strictly linear, the security mechanisms could be implemented similarly as well. For example, threat analysis could be initially performed early in the SDLC and reviewed during each sprint, or when important security-related sprint backlog items are implemented.
3. The model offers guidance and proposes practicable tools and methods for all phases of Agile software development from inception to delivery phase of the project.
4. The model greatly relies on mature level of adoption of endogenous control mechanisms in Agile Methods. This way security is adaptable to rapid changes in business environment because it is directed by the incremental Agile development process instead of a rigid pre-determined set of rules.

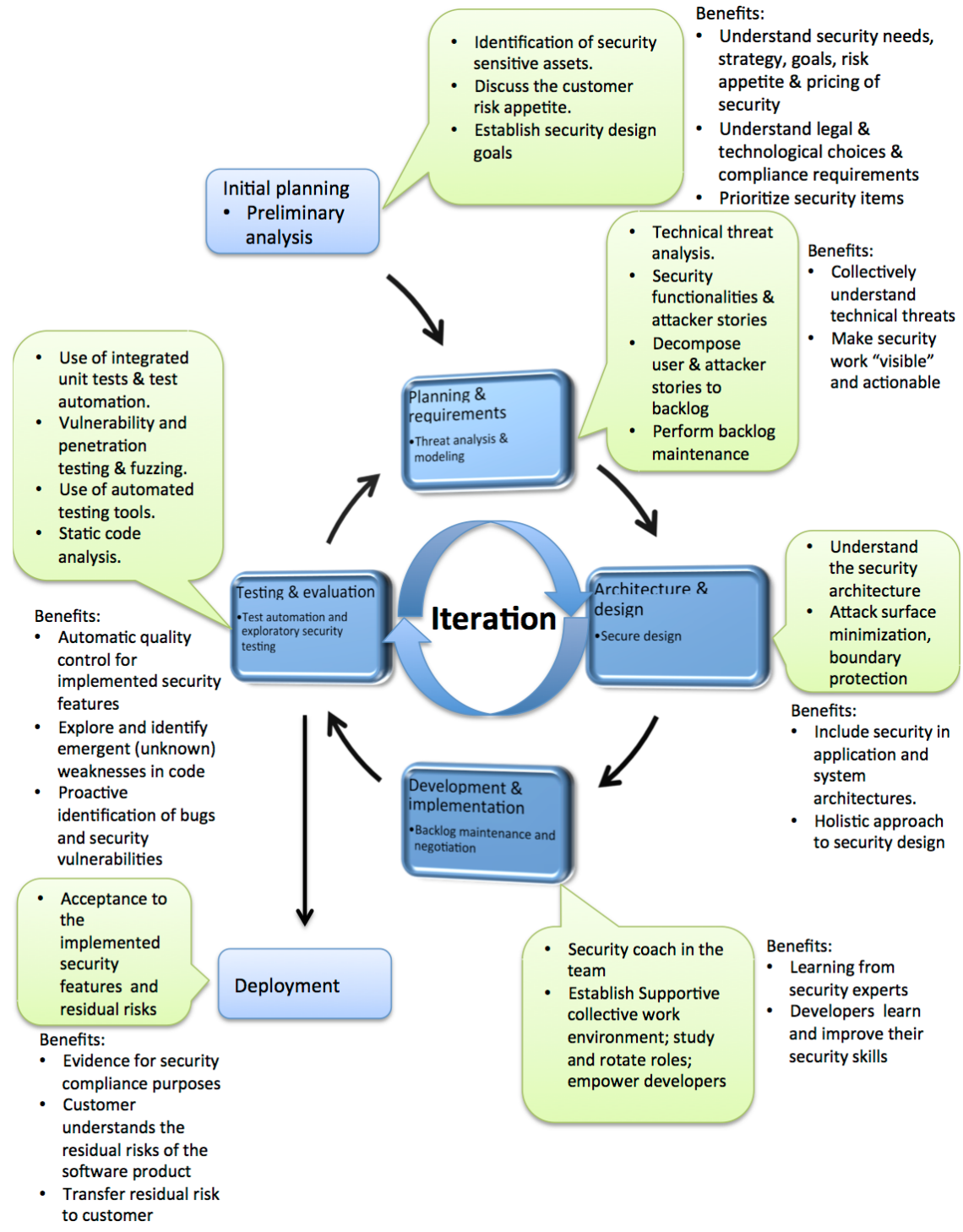


Figure 15: The Agile Security Model

9 Conclusions

The main goal of this research was to construct an Agile Security Model that would be universally applicable in most organizations using Agile Methods. The construction of the security model was a 2-year effort that involved working in Silverskin Information Security with several customer projects. We were able to utilize basic theory of Agile software development in real-life cases and tested parts of the security model on them. Finally, a holistic security model was created that addressed the most important security mechanisms that we believe to be critical for developing secure software. The model's closest relative is the Agile SDLC from Microsoft (2009). The difference between the two is that Microsoft's model proposes a wide variety of developer tasks to be finished at the end of each sprint (or bucket, as Microsoft calls their development cycles). In contrast our model supposes that these tasks are driven by business demands and they should be taken to backlog as development tasks just like all other tasks.

Our observation from the real world is that organizations have implemented some parts of the mechanisms in the security model while excluding some others. Failing to implement some of the critical mechanisms has proven to create software vulnerabilities and security problems. Those problems come in various degrees. In some instances the problems manifest themselves in the form of vulnerabilities in the applications, and in other we have observed that the problems arise from failure in business-level requirements that may ruin the whole software project. The research by WASC (2008), Silverskin (2012) and Ponemon Institute (2013) have shown that inadequate development practices can lead to vulnerable software. Our Agile Security Model has been tested in few real-life cases and resulted to very secure outcomes from software development. The software projects that performed their development in accordance with the model did extremely well when final penetration testing was performed against the applications.

Therefore, the efficient control of software security in Agile requires that all critical components in the software development process are taken into consideration. Failure to do so will most likely lead to security problems. Our Agile Security Model approaches the Agile software security from a holistic perspective and addresses most of these problems. The model should be applicable to most organizations in Agile software development business, and should not burden the developers with ineffective security tasks that do not make sense to them. The model can be applied by applying parts of it to the software development lifecycle based on the level of required formalism and amount of desired security. As opposed to many other models (Microsoft, Cisco), the model does not expect the organization to implement every security mechanism, nor does it skip addressing the process view of the development (OpenSAMM).

We have also gained business benefits from the development and application of the model in real world customer interactions at Silverskin. The model has increased our knowledge about the topic, and provided us with a framework that we can re-use in the future interactions. We urge the security community and the organizations in the Agile development business to test the model in practice and develop it further to more thoroughly serve the goal of creating secure software.

9.1 Reflection About the Success of the Research

According to YAMK (Method forum of Universities of Applied Sciences), there are several criteria for assessing successfulness of a research. The three main factors to be assessed are:

1. Practical usefulness
2. Transferability and applicability of the results under various conditions
3. The novelty of the results

The model has been tested in various customer engagements in Silverskin. Implementation of the security mechanisms in the model has resulted to secure outcomes from software development. This has resulted to new software security consultancy services. The CEO of the company commented: "the model presented by this research enables us to develop new ways of offering the services to our customers". The model was tested with several different customers and was reviewed by a team of Agile software architects in a software security seminar on October 8th 2013. When questioned about the transferability the architects believed that the model could well be implemented on their own organizations. The model uses a novel approach of using endogenous process controls to include security in development instead of exogenouse controls. This is a novel approach to formulating an entire Agile process model. A close relative to this model is the Microsoft Agile SDLC.

Successulness of the project can be also assessed based on the following factors:

Research goals	**Assessment**
Does the research focus on a problem of the real world that can be solved? (Anttila, 2007)	The problem of insecure software is real as demonstrated by WASC (2008) and the study performed by Silverskin (2012). The Agile Methodologies don't include adequate security mechanisms (Beznosov, 2004).
Did the solution provide a totally new or improved artefact? (Anttila, 2007)	Our Agile Security Model primarily uses endogenous process controls instead of exogenouse controls. This is a novel approach to formulating an entire secure Agile process model. A close relative to this model is the Microsoft Agile SDLC.
Was the problem generally deemed to be necessary to be solved? (Anttila, 2007)	Research by several authors like Beznosov (2004), Peeters (2005), Siponen, Baskerville & Kuivalainen (2005) has indicated that Agile Software Development requires integrating security practices into the development.
Is the artefact useful and can it be reused in other instances besides the research project itself? (Anttila, 2007)	The Agile Security Model was constructed so that it can be generally applied under various conditions. It is believed that the model could be used by a software development company, or by a security consultancy company likewise.
Is the artifact constructed so that it can be conceptualized and	The Agile Security Model is an artifact based on general-concepts in Agile software development and from the security industry. It is likely that anyone with basic un-

understood in general terms? Is the conceptualization of the artefact understandable? (Anttila, 2007)	derstanding from these fields of business would be able to undestand and use the model.
Has the artefact been tested in practice, and did the solution result to improved results? (Anttila, 2007)	We tested the model with several organizations who use Agile development methods. Those organizations who invested more effort on security fared better when their software was audited prior to deployment than those who had not integrated security activities into their development processes.

Table 19: Assessment of the successfulness of the research.

In summary, the research project has been a success. We have gained business benefit from the research and learned as an organization to improve our services. The model provides a novel approach for secure software development, and can be used generally in many kinds of organizations. However, more empirical testing work can always be executed for fine-tuning and developing the model further. In principle, any model can never be quite perfect since all models are approximations of the phenomena in the real world. The following table summarizes the goals and results of the research:

Sources

Agile Manifesto. Available online at http://www.agilemanifesto.org

Anttila, P. 2007. Realistinen evaluaatio ja tuloksellinen kehittämistyö. Hamina, Akatiimi

Asthana, V., Tarandach, I., O'Donoghue, N., Sullivan, B., & Saario, M. 2012. Practical Security Stories and Security Tasks for Agile Development Environments. SAFECode, Software Assurance Forum for Excellence in Code. http://www.safecode.org/publications/SAFECode_Agile_Dev_Security0712.pdf

Baskerville, R. & Siponen, M. 2002. An information security meta-policy for emergent organizations. Published in Logistics Information Management vol. 15. Available in http://www.emeraldinsight.com/0957-6053.htm

Beznosov, K & Kruchten, P. 2004-2006. Towards Agile Security Assurance (presentation slides). Laboratory of education and research in secure systems engineering. University of British Columbia. http://lersse-dl.ece.ubc.ca/record/117/files/117.pdf

Beznosov, K. & Kruchten, P. 2004. Towards Agile Security Assurance. Proceedings of The New Security Paradigms Workshop, White Point Beach Resort, Nova Scotia, 20-23 September 2004. http://www.nspw.org/papers/2004/nspw2004-beznosov.pdf

Beznosov, K. 2003. Extreme security engineering: on employing xp practices to achieve "good enough security" without defining it. In First ACM Workshop on Business Driven Security Engineering (BizSec), Faiffax, VA, USA. Electrical and Computer Engineering University of British Columbia. http://lersse-dl.ece.ubc.ca/record/43/files/43.pdf

Beznosov, K., Boden, M., Boström, G., Kruchten, P. & Wäyrynen, J. 2006. Extending XP practices to Support Security Requirements Engineering. SESS'06, May 20–21, 2006, Shanghai, China. http://www.irisa.fr/lande/lande/icse-proceedings/sess/p11.pdf

Baskerville, R. 2004. Agile security for information warfare: a call for research. Georgia State University, 35 Broad Street NW, Atlanta, Georgia 30302, USA

Binder, R. V. 2004. Testing object-oriented systems: models, patterns, and tools. Addison-Wesley publishing. ISBN 0201809389.

Boehm, B.W. 1983. Software Engineering Economics. Software Information Systems Division, TRW Defense Systems Group, CA, USA. Available online at:

http://csse.usc.edu/csse/TECHRPTS/1984/usccse84-500/usccse84-500.pdf

CIO Magazine. 2004. 100 most agile companies honored. Available online at: http://www.cio.com/article/368313/100_Most_Agile_Companies_Honored

Crnkovic, G.D. 2010. Constructive Research and Info-Computational Knowledge Generation. School of Innovation, Design and Engineering, Computer Science Laboratory, Mälardalen University, Sweden. Available online at http://www.springerlink.com

Cisco Secure Development Lifecycle. Available in: http://www.cisco.com/web/about/security/cspo/csdl/index.html

Cunningham, W. 1992. The WyCash Portfolio Management System. Available online at http://c2.com/doc/oopsla92.html

De Lucia, A., Ferrucci, F. & Tortora, G. 2008. Emerging Methods, Technologies, and Process Management in Software Engineering. Wiley, Hoboken, NJ, USA. pISBN: 9780470085714

Dhalla, R. 2009. Reputational risk: implications for organizational strategy. Department of Business, College of Management and Economics. University of Guelph, Ontario Canada.

École Polytechnique de Montréal. Unified Process for EDUcation Glossary. http://www.upedu.org/process/glossary.htm

Financial Services Roundtable. 2012. BITS Software Assurance Framework. Washington DC, USA. www.bits.org

Firesmith, D. G. 2003. Common Concepts Underlying Safety, Security, and Survivability Engineering (CMU/SEI-2003-TN-033). Software Engineering Institute, Carnegie Mellon University.
http://www.sei.cmu.edu/library/abstracts/reports/03tn033.cfm

Hernan, S., Lambert S., Ostwald T., & Shostack A. 2006. Uncover Security Design Flaws Using The STRIDE Approach. MSDN Library.
http://msdn.microsoft.com/en-us/magazine/cc163519.aspx

Layton, M. C. 2012. Agile Project Management for Dummies: What is an Agile sprint backlog? Excerpt available from http://www.dummies.com/how-to/content/what-is-an-agile-sprint-backlog.html

Lukka Kari. 2001. Konstruktiivinen tutkimusote. Luettu 28.1.2011. www.metodix.com

Malone T.W. & Crowston K., 1990. What is Coordination Theory and How Can It Help Design Cooperative Work Systems. ACM conference on Computer supported cooperative work, Los Angeles, CA, USA.

Malone T.W. & Crowston K., 1994. The Interdisciplinary Study of Coordination. ACM Computing Surveys, Vol. 26, No. 1. Available online at http://www.cs.unicam.it/merelli/Calcolo/malone.pdf

Mead, N. R., Hough E. D. & Stehney, T. R. II. 2005. Security Quality Requirements Engineering (SQUARE) Methodology. CMU/SEI-2005-TR-009. Engineering Institute, Carnegie Mellon University.

Mead, N.R. 2010. Security Requirements Reusability and the SQUARE Methodology. CMU/SEI-2010-TN-027. Engineering Institute, Carnegie Mellon University.

Microsoft. 2005. The STRIDE Threat Model. MSDN Library. http://msdn.microsoft.com/en-us/library/ee823878(CS.20).aspx (2005).

Microsoft Corporation. 2009. Security Development Lifecycle for Agile Development, Version 1.0. Available at http://www.microsoft.com/security/sdl/discover/sdlagile.aspx and www.blackhat.com/presentations/bh-dc-10/Sullivan_Bryan/BlackHat-DC-2010-Sullivan-SDL-Agile-wp.pdf

MITRE Corporation. Common Attack Pattern Enumeration and Classification (CAPEC), Community Knowledge Resource for Building Secure Software, CAPEC-1000: Mechanism of Attack. Available in: http://capec.mitre.org/data/graphs/1000.html#Definition

Noopur Davis. 2005. Secure Software Development Life Cycle Processes: A Technology Scouting Report (CMU/SEI-2005-TN-024). Software Engineering Institute, Carnegie Mellon University

OpenSAMM. 2013. Software Assurance Maturity Model. Available online at http://www.opensamm.org/

Pauli, J. & Xu, D. 2004. Misuse Case-Based Design and Analysis of Secure Software Architecture. Department of Computer Science North Dakota State University. Available in electronic format in: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.4188&rep=rep1&type=pdf

Peeters J. 2005. Agile Security Requirements Engineering. Presented at the Symposium on Requirements Engineering for Information Security. http://secappdev.org/handouts/2008/abuser%20stories.pdf

Ponemon Institute & Security Innovation. 2013. The state of application security, a research study. https://www.securityinnovation.com/uploads/ponemon-state-of-application-security-maturity.pdf

Saleh, K. A. 2009. Software Engineering. J. Ross Publishing Inc. Ft. Lauderdale, FL, USA. eISBN: 9781604276749

Scrum Alliance. 2007. Glossary of scrum terms. http://www.scrumalliance.org/community/articles/2007/march/glossary-of-scrum-terms#1125

Siddhu, D.P. & Leung, T-K. 1989. Formal methods for protocol testing: a detailed study. IEEE Transactions on Software Engineering (413-426).

Sillitti, A. & Succi, G. 2008. Foundations of Agile Methods. Free University of Bolzano-Bozen, Italy.

SANS Institute. 2013. Advanced Penetration Testing, Exploits and Ethical Hacking: Python, Scapy and Fuzzing. Security 660: Course study book 3. SANS Institute, USA.

Silverskin Information Security LLC. 2012. Causes and effects of web application security vulnerabilities. A research paper that combines results of 130 web application security audits.

Sindre, G. & Opdahl A.L. 2002. Eliciting security requirements with misuse cases. Springer-Verlag London Limited 2004. Available online at http://perceval.gannon.edu/xu001/teaching/shared/re_eng/termpaper/readingList/ElicitingSecReq_MisuseCases.pdf

Siponen, M., Baskerville, R. & Kuivalainen, T. 2005. Integrating Security into Agile Development Methods. University of Oulu & Georgia State University.

Strode, D. 2006. Agile methods: a comparative analysis. Faculty of Business and Information Technology. Whitireia Community Polytechnic. Porirua, New Zealand

Särs, C. 2010. Agile Security: The Devil's Advocate. F-secure. Online presentation, available from http://confluence.agilefinland.com/download/attachments/6848543/AgileSecurityDevilsAdvocate_20100406.pdf?version=2&modificationDate=1286273115000

Standish Group. 1994. CHAOS Report. Available online at: http://www.standishgroup.com/sample_research/chaos_1994_1.php

The Free Dictionary. The definition of Information Security. Available online at: http://www.thefreedictionary.com/information+security

Threat Classification Development View. 2008. http://projects.webappsec.org/w/page/13246969/Threat%20Classification%20Development%20View

University of Virginia, Center for Risk Management for Engineering Systems. Web page, ”Risk Defined”. http://www.sys.virginia.edu/risk/riskdefined.html

Valtionhallinnon tietoturvallisuuden johtoryhmä. 2013. VAHTI 1/2013 sovelluskehityksen tietoturvaohje. Valtionvarainministeriö. Suomen Yliopistopaino Oy. ISBN 978-952-251-418-9

Virtanen Aila 2006. Konstruktiivinen tutkimusote. Miten koulutus ja elinkeinoelämän odotukset kohtaavat ammattikorkeakoulun opinnäytetyössä. Ammattikasvatuksen aikakauskirja 1/2006, 46-52.

Vähä-Sipilä, A. 2011. Software security in Agile product management. http://www.fokkusu.fi/agile-security/

WASC Security Glossary. http://www.webappsec.org/projects/glossary/

WASC Threat Classification v2.0. 2011. http://projects.webappsec.org/w/page/13246978/Threat%20Classification

WASC Web Application Security Statistics. 2008. http://projects.webappsec.org/w/page/13246989/Web%20Application%20Security%20Statistics

WASC Web Hacking Incident Database. http://projects.webappsec.org/w/page/13246995/Web-Hacking-Incident-Database

Ylemmän AMK:n metodifoorumi (YAMK). Kehittämishankkeen onnistumisen kriteerit. http://www2.amk.fi/digma.fi/www.amk.fi/opintojaksot/0709019/1193463890749/1193464185783/1194413827887/1194415395853.html

Ylemmän AMK:n metodifoorumi (YAMK). Tulosten ja työn hyödynnettävyys. http://www2.amk.fi/digma.fi/www.amk.fi/opintojaksot/0709019/1193463890749/1193464185783/1194413827887/1194415412494.html

Figures

Figure 1: Critical findings per audit category (Silverskin, 2012) 21
Figure 2: Distribution of findings in audit categories per origin (Silverskin, 2012) 22
Figure 3: Outcomes of software vulnerabilities (WASC web hacking incident database) 24
Figure 4: Constituents of the constructed Agile security model. 30
Figure 5: Construction phases of the Agile Security Model 31
Figure 6: Traditional waterfall software development model (Sillitti & Succi, 2008) 48
Figure 7: Iterative Agile software development process model (Sillitti & Succi, 2008) 49
Figure 8: Agile pivot when business targets change (Vähä-Sipilä, 2011) 50
Figure 9: Agile product management funnel (Vähä-Sipilä, 2011) 52
Figure 10: Problem-based construction process for the Agile Security Model. 53
Figure 11: Structure of a user story (De Lucia & al. 2008) 72
Figure 12: Relation of positive and negative security features. 75
Figure 13: Misuse based threat modeling (Pauli & Xu, 2004) 76
Figure 14: A dynamic view of application security architecture. 83
Figure 15: The Agile Security Model 93
Figure 16: Cisco SDLC model (Cisco) 105
Figure 17: Microsoft SDLC, linear presentation (Microsoft) 106
Figure 18: Microsoft Agile SDLC, iterative presentation (Microsoft, 2009) 106
Figure 19: OpenSAMM Software Assurance Maturity Model (OpenSAMM) ... 106
Figure 20: Information model for security engineering (Firesmith, 2003, p. 33) 111

Tables

Table 1: Main causes of software development project failure (Standish Group, 1994)..... 14
Table 2: Compatibility analysis of software assurance methods (Beznosov & Kruchten, 2004). 18
Table 3: WASC view on development of threat classification. 23
Table 4: Silverskin's undertakings where the constructed security model was tested. 31
Table 5: Literature analysis technique. 34
Table 6: Comparison of two different software development literature sources. 36
Table 7: Analysis of internal and external process controls. 44
Table 8: Exogenous and endogenous controls in extreme programming (Sillitti & Succi, 2008) 46
Table 9: Sequential, shared resource and common output controls (Malone & Crawston, 1990)..... 47
Table 10: Core principles, problems and possible impacts in Agile software development..... 54
Table 11: Critical success factors in secure software development (N=16 sources from literature) 58
Table 12: Analysis #1 of perceived problems and proposed control mechanisms for Agile. 61
Table 13: Analysis #2 of perceived problems and proposed control mechanisms for Agile. 62
Table 14: Analysis #3 of perceived problems and proposed control mechanisms for Agile 63
Table 15: Analysis #4 of perceived problems and proposed control mechanisms for Agile. 64
Table 16: Analysis #5 of perceived problems and proposed control mechanisms for Agile. 65
Table 17: Example of a non-functional security requirement (CAPEC-1000)74
Table 18: Example of decomposing attacker stories to backlog..... 79
Table 20: Assessment of the successfulness of the research..... 97

Attachment 1. Related Security Models

The following software development security lifecycle (SDLC) models are related to the Agile Security Model presented in this research. It is noteworthy that most of the models are linear (Figures 16 & 17) and resemble the waterfall ideology, although their constituents are similar to the ones in our Agile Security Model. Microsoft has also created an iterative representation of their agile development model (figure 18). Microsoft's model is truly and attempt to integrate security into the Agile. However, it mostly imposes several exogenous controls to the sprints, 'buckets' and 'one-time activities' in the development process (e.g. each sprint includes mandatory security activities). The shortcoming is that all of the activities in this model may not be applicable to all kinds of software development organizations, especially those who have fewer resources in their use than Microsoft does. OpenSAMM, on the other hand does not tie it's SDL model to software development process models at all, but rather proposes controls independently from the development methodology. Therefore, it is possible that all of the controls proposed in OpenSAMM may not be compatible with the different development process models, or they may seem external to the actual development work.

Figure 16: Cisco SDLC model (Cisco)

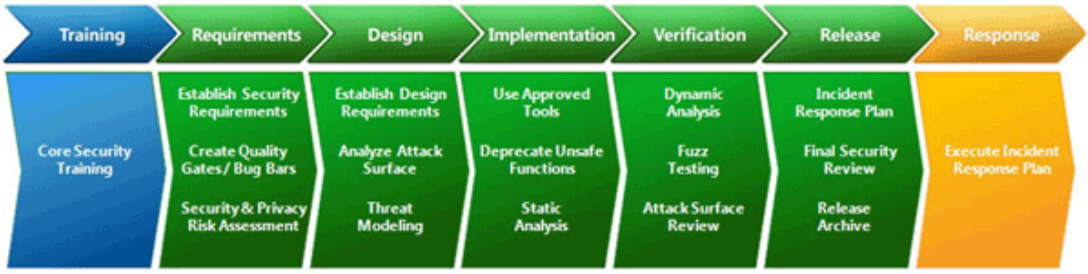


Figure 17: Microsoft SDLC, linear presentation (Microsoft)

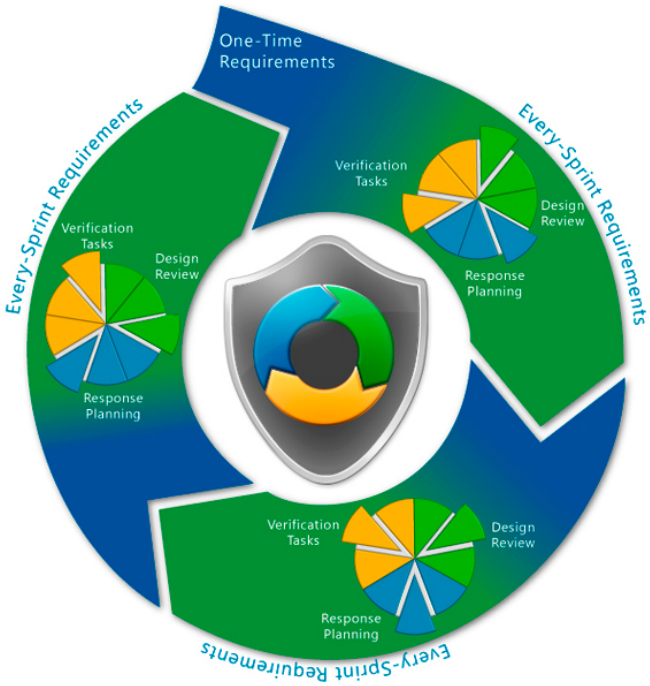


Figure 18: Microsoft Agile SDLC, iterative presentation (Microsoft, 2009)

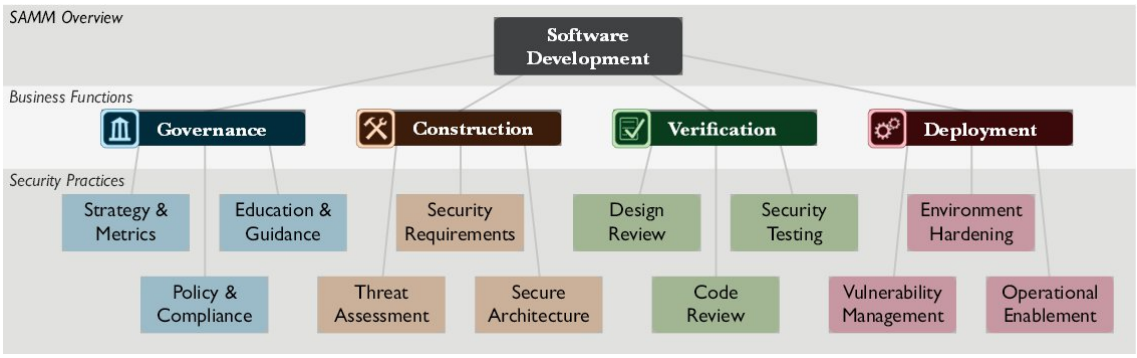


Figure 19: OpenSAMM Software Assurance Maturity Model (OpenSAMM)

Attachment 2. Other Related Concepts

Agile Manifesto: the Agile Manifesto summarizes the common philosophy and approach shared by of all the Agile development Methods.

Application Development: software development (also known as application development, software design, designing software, software application development, enterprise application development, or platform development) is the development of a software product (Wikipedia).

Authentication: the process of verifying the identity or location of a user, service or application. (WASC Glossary)

Assurance: assurance in software engineering is defined as the level of confidence that software does not contain vulnerabilities and that it functions in the intended manner.

Brute Force: is an automated iterative process that aims to find the "secret" protecting a system, for example a password or a secret encryption key. (WASC Glossary)

Buffer Overflow: is an exploitation technique that alters the flow of an application by overwriting parts of memory. Specifically this is accomplished by first over-filling and overwriting the buffers in memory and then overwriting the return pointer to the memory stack. This allows running of arbitrary malicious code. (WASC Glossary)

Cookie: small amount of data sent by the web server, to a web client, which can be stored and retrieved at a later time. Typically cookies are used to keep track of a user's state (e.g. web application sessions) as they traverse a web site. (WASC Glossary)

Control: is, in the context of security engineering, synonymous to Security Mechanism. In software development context, a control is a means to control the flow of the development process via some kind of a shared common resource (Malone & Crowston, 1990).

Cross-Site Scripting (XSS): an attack technique that forces a web site to echo client-supplied data, which executes in a user's web browser. This allows the attacker to have access to all web browser content of the victim (cookies, history, application version, etc). In worst scenario, the attacker may be able to stage further attacks against the victim and elevate his privileges to total system-level compromise. (WASC Glossary)

Denial of Service: or Distributed Denial of Service (DoS or DDoS) are attack techniques that consume the resources (CPU, memory, network bandwidth / etc) of the target. The intent is to render normal use of the target impossible. (WASC Glossary)

Fuzzing: is a software testing technique that involves providing invalid, unexpected, or random data to the inputs of a computer program. (Wikipedia).

Information security: the protection of information and information systems against unauthorized access or modification of information, whether in storage, processing, or transit, and against denial of service to authorized users (The free dictionary).

Risk: generally speaking risk is usually defined as a measure of the probability and severity of its adverse effects. Various definitions of risk are in existence depending on the type of risk. For instance in security engineering, risk is the result of a threat with adverse effects to a vulnerable system (University of Virginia).

Risk appetite: the level of risk that an organization is prepared to accept, before action is deemed necessary to reduce it (Wikipedia).

SQL Injection: an attack technique used to exploit web sites by altering backend SQL statements through manipulating application input. (WASC Glossary)

Technical debt: according to Vähä-Sipilä, "…Not addressing quality issues right away is known as technical debt. Technical debt borrows time from the future: some corners can be cut now, but someone should spend time later making sure those things are mopped up later. There are various forms of technical debt. Security debt is very close to quality debt, which can be caused … by lack of testing due to a business owner ordering a premature delivery. (Vähä-Sipilä, 2011 and Cunningham, W. 1992)

Security Development Lifecycle (SDL): "…is a software development process that helps developers build more secure software and address security compliance requirements while reducing development cost" (Microsoft).

Software requirement: a specification of an externally observable behavior of the system; for example inputs to the system, outputs from the system, functions of the system, attributes of the system, or attributes of the system environment. (UPEDU)

Security testing: is a way to identify security–related errors in the software. Generally speaking there are three security testing approaches: black box, white box and grey box. In black box approach, nothing is known about the tested application prior to testing. In white box, all information about the tested application is made available. The grey box approach is the mixture of the two where some features of the target are known, but not everything. (Saleh, 2009).

Security goal: Is a statement that explains the importance of reaching a target level of security (Firesmith, 2003).

Static code analysis: is the analysis of computer software that is performed without actually executing programs (Wikipedia).

Threat: is the intent and capability to adversely affect (cause harm or damage to) the system by adversely changing its states (University of Virginia).

User story: (sometimes referred as Use Case) is a short description in common language that explains what a user needs to do as part of his job. User stories are used in Agile software development as the basis for defining the functionalities the developed system needs to provide (Vähä-Sipilä, 2011)

Vulnerability: is the manifestation of the inherent states of the system (e.g., physical, technical, organizational, cultural) that can be exploited to adversely affect (cause harm or damage to) that system (University of Virginia).

Waterfall model: "…is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation, and Maintenance" (Wikipedia).

Web Application: a software application, executed by a web server, which responds to dynamic web page requests over HTTP. (WASC Glossary)

Web Application Vulnerability Scanner: An automated security program that searches for software vulnerabilities within web applications. (WASC Glossary)

Attachment 3. Information Model of Security Engineering

The following conceptual information model for security engineering (Firestmith, 2003) illustrates the interrelations of several of the security engineering concepts used in this research (Figure 19).

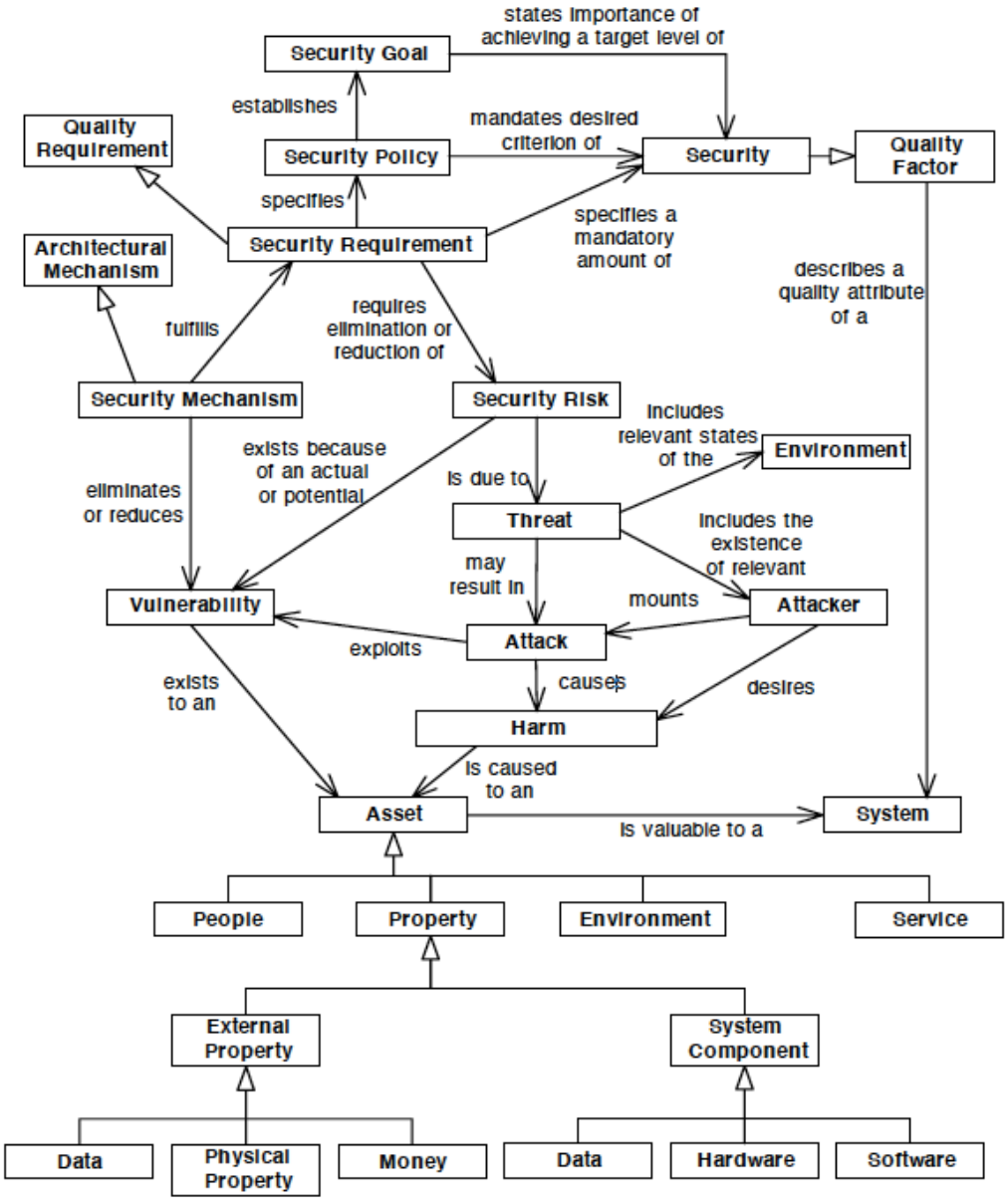


Figure 20: Information model for security engineering (Firesmith, 2003, p. 33)