



Patrik Maier, BSc

Integrating Web Application Security into the Agile Software Development Process

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Ph.D. Roderick Bloem

Institute of Applied Information Processing and Communications (IAIK)

Dr. Zhendong Ma

Austrian Institute of Technology (AIT)

Graz, 25 Jan 2017

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

Abstract

Agile methods, such as Scrum and Extreme Programming, have started to replace models like waterfall, V-model and prototyping. One of the reasons was the old models' inability to respond to constantly changing business requirements. Agile development is based on short iteration cycles which allow it to quickly respond to changes in business requirements. The demands for web application security and information security in general are rising due to the increasing number of different organizations suffering from security breaches of their IT infrastructure and applications. However, most of the existing security processes, frameworks and tools are based on traditional processes. This work presents a proposal of a secure Scrum process for developing secure web applications. We suggest high level security activities which extend the Scrum development process.

First, we analysed how the security engineering practices of the Security Engineering Capability Maturity Model (SSE-CMM) could be integrated into Scrum. The next step was to select the most agile activities and tools of well-established security engineering processes such as Microsoft SDL and NIST 800-64 and to introduce them into Scrum. In order to rate all activities relatively to each other regarding their agility we used an existing method and adjusted it slightly. At last, we evaluated the agility of this new process, which security level it provides and how much more time is needed to complete each of its activities in comparison to Scrum. In order to measure its agility and additional time effort, we conducted a survey in which we asked employees of a company which is developing JAVA EE applications using Scrum how agile each activity is and how long it takes to complete each activity. For evaluating our process' provided security, we evaluated this new process with OWASP-SAMM and calculated maturity levels of several categories.

On the one hand, according to our analysis, this process retains a great amount of the flexibility and efficiency provided by an agile development process. On the other hand it supports the development of secure web applications. By "secure" we mean that applications are not vulnerable to the OWASP Top 10 risks and the process fulfils all goals of the SSE-CMM.

Kurzfassung

Agile Methoden, wie Scrum und Extreme Programming (XP), haben damit begonnen die traditionellen Softwareentwicklungsprozesse (Vorgehensmodelle) wie das Wasserfall-Modell, V-Modell und Prototyping abzulösen. Einer der Gründe dafür war, dass die traditionellen Vorgehensmodelle nicht mit rasch ändernden Kundenanforderungen an die Software umgehen können. Im Gegensatz zu den traditionellen Vorgehensmodellen wird Software in agiler Softwareentwicklung in kleinen Iterationszyklen entwickelt, wodurch schnell auf Kundenanforderungsänderungen reagiert werden kann. Die Forderung nach "sicheren" Webapplikationen und Informationssicherheit im Allgemeinen steigt, da immer mehr Organisationen von Cyberattacken betroffen sind. Dennoch sind viele der bestehenden Sicherheitsprozesse, Frameworks und Tools für die traditionellen Softwareentwicklungsprozesse entworfen worden. Diese Masterarbeit präsentiert einen sicheren Scrum-Entwicklungsprozessvorschlag für die Entwicklung von sicheren Webapplikationen. Um dieses Ziel zu erreichen integrieren wir so genannte "Sicherheitsaktivitäten" in einen auf Scrum basierenden Softwareentwicklungsprozess.

Zuerst analysierten wir ob Sicherheitspraktiken von SSE-CMM in Scrum integriert werden können. Als nächstes untersuchten wir die Agilität von Sicherheitsaktivitäten und Tools von vorhanden sicheren Softwareentwicklungsprozessen wie Microsoft SDL und NIST 800-64 und integrierten die agilsten in Scrum. Um die Agilität der einzelnen Sicherheitsaktivitäten zu evaluieren verwendeten wir eine in der Literatur bereits existierende Methode und modifizierten sie ein wenig. Als letzten Schritt evaluieren wir die Agilität des von uns entwickelten sicheren auf Scrum basierenden Softwareentwicklungsprozessvorschlags, wieviel Sicherheit dieser Prozessvorschlag gewährleistet und wieviel Zeit für die Durchführung der neu eingeführten Sicherheitsaktivitäten schätzungsweise benötigt wird. Für die Evaluierung führten wir in einem Unternehmen, welches JAVA EE Applikationen mittels Scrum entwickelt, eine schriftliche Befragung mit Hilfe eines Fragebogens durch. Wir fragten einige Mitarbeiter wie agil jede einzelne Sicherheitsaktivität unseres sicheren Softwareentwicklungsprozesses ist und wie hoch sie den zusätzlichen Zeitaufwand für die Durchführung der einzelnen Sicherheitsaktivitäten schätzen. Um herauszufinden wieviel Sicherheit dieser Prozessvorschlag gewährleistet, führten wir ein Assessment mit OWASP-SAMM durch.

Laut unserer Evaluierung behält unser Softwareentwicklungsprozessvorschlag einen großen Teil der Effizienz und Flexibilität eines agilen Softwareentwicklungsprozesses und bietet Unterstützung für die Entwicklung von sicheren Webapplikationen. Bei "sicher" meinen wir, dass die Applikation nicht verwundbar gegen die OWASP Top 10 ist und alle Ziele des SSE-CMM erfüllt.

Contents

Contents	vi
List of Figures	viii
List of Tables	ix
1 Introduction	1
1.1 Background	1
1.2 Problem definition	2
1.3 Intent of this Thesis	2
1.3.1 What has Already Been Done in Literature	2
1.3.2 Our contributions	3
1.4 Our approach in further detail	3
1.4.1 Integrating SSE-CMM practices into Scrum	4
1.4.2 Evaluation of activities and tools of existing SE-processes	4
1.4.2.1 Brief overview of the method and our modifications	5
1.4.2.2 Results	5
1.4.3 Agile-friendly risk analysis	6
1.4.4 Our evaluation	7
1.4.4.1 Results	7
1.5 Structure of the thesis and the methodology	7
2 Related work	9
2.1 Analysis of Scrum	9
2.2 Security requirements in agile development processes	10
2.3 Agile security engineering activities	11
2.4 Agile threat modeling/risk assessment	13
2.5 Summary	13

3 Preliminaries	15
3.1 Agile Manifesto	15
3.2 Scrum	16
3.2.1 Definition of Done	16
3.2.2 Scrum Events	16
3.2.3 Scrum Skeleton	17
3.2.4 Further agile practices	18
3.2.4.1 User Stories	18
3.2.4.2 Definition of Ready	18
3.2.4.3 Scrum Release Planning	18
3.3 OWASP Top 10	18
3.4 Preliminaries regarding risk assessment and threat modeling	19
3.4.1 STRIDE	21
3.4.2 Application Security Frame (ASF)	21
3.4.3 DREAD	21
3.4.4 Threat Modeling	22
3.4.5 Microsoft’s Threat Modeling method	22
3.4.6 OWASP Application Threat Modeling	24
3.4.6.1 Decompose the Application	24
3.4.6.2 Determine and Rank Threats	25
3.4.6.3 Countermeasure Identification and Mitigation Strategies	25
3.4.7 Risk assessment	26
3.4.7.1 NIST SP 800-30 Guide for Conducting Risk Assessments	26
3.5 Common Criteria	28
3.6 System Security Engineering Capability Maturity Model	28
3.7 Preliminaries regarding software testing	33
3.7.1 Security Testing	33
3.7.2 Black-box web vulnerability scanner	33
3.7.3 Penetration testing	33
3.7.4 Static code analysis	34
3.7.5 Dynamic code analysis	34
3.7.6 Red team testing	34
3.7.7 Fuzz Testing	35
3.8 Security Engineering processes	35
3.8.1 NIST 800-64	35
3.8.2 Microsoft Secure Development Lifecycle (Microsoft SDL)	37
3.8.3 Security Requirements Engineering Process (SREP)	38
3.8.4 Cigital Touchpoint	39

4	Analysis of Scrum	41
4.1	SSE-CMM's Processes	41
4.1.1	Risk Process	41
4.1.2	Engineering Process	42
4.1.2.1	PA 09 ("Provide Security Input") and PA 10 ("Specify Security Needs")	43
4.1.2.2	PA 07: Coordination of Security	45
4.1.3	Assurance Process	45
4.1.3.1	PA 06 ("Build assurance argument")	46
4.1.3.2	PA 11: Verify and Validate Security	46
4.1.4	Summary of this Chapter	48
5	Analysis of Existing Well-Established Security Activities and Tools	51
5.1	Our approach	52
5.1.1	Our interpretation of agile features	52
5.1.2	Our metric	53
5.1.3	Validation of results	53
5.2	Activities and tools assessment	56
5.2.1	Microsoft SDL	56
5.2.1.1	Security Requirement Analysis	56
5.2.1.2	Role Matrix	56
5.2.1.3	Design Requirements	56
5.2.1.4	Cost analysis (By conducting a Product Risk Assessment)	56
5.2.1.5	Threat Modeling	57
5.2.1.6	Attack Surface Analysis / Reduction	57
5.2.1.7	Static Analysis	57
5.2.1.8	Dynamic Analysis	57
5.2.1.9	Fuzz Testing	57
5.2.1.10	Code Review	57
5.2.2	Cigital Touchpoint	57
5.2.2.1	Security Requirement Analysis	57
5.2.2.2	Abuse Cases	57
5.2.2.3	Architectural Risk Analysis	58
5.2.2.4	Manual or semi-automatic Penetration Testing, Automatic Penetration Testing, Static Analysis, and Red Team Testing	58
5.2.2.5	Risk Based Testing	58
5.2.3	SREP	58
5.2.3.1	Security Requirement Analysis	58
5.2.3.2	Agree on Definitions	59
5.2.3.3	Risk Analyses	59
5.2.3.4	Critical Assets	59
5.2.3.5	Identify Threats and Develop Artifacts	59
5.2.3.6	Requirement Inspection	60
5.2.3.7	Repository Improvement	60

5.2.4	NIST 800-64	60
5.2.4.1	Initiate Project Security Planning	60
5.2.4.2	Categorize Information System	60
5.2.4.3	Assess Business Impact	61
5.2.4.4	Assess Privacy Impact	61
5.2.4.5	Assess Risk to System	61
5.2.4.6	Select and Document Security Controls	61
5.2.4.7	Design security architecture	62
5.2.4.8	Conduct Testing	62
5.3	Results of the evaluation	63
5.4	Assign security activities to SSE-CMM's PAs	65
5.4.1	Summary of the evaluation results	65
5.4.2	Assign security activities to SSE-CMM's PAs	66
5.4.3	Integrating security activities into Scrum	68
5.4.3.1	PA 02–05	68
5.4.3.2	PA 06	69
5.4.3.3	PA 09	69
5.4.3.4	PA 10	70
5.4.3.5	PA 11	70
6	Agile Risk Analysis	73
6.1	Motivation	73
6.2	Intent of this chapter	73
6.3	Determine External Dependencies and Security Assumption (External Dependencies)	75
6.4	Identify Assets	76
6.5	Identify Trust Levels	78
6.6	Determine entry points	80
6.7	Identify Threats	82
6.8	Ranking of Threats with DREAD	83
6.9	Mitigation Strategies	83
7	Proposal for an Agile Secure Process (Secure Scrum)	85
7.1	High level description	85
7.1.1	Preliminaries	85
7.1.1.1	Documentation of Security Requirements	85
7.1.1.2	Required security training of developers and additional Scrum roles	86
7.1.2	High-level overview of the proposal	86
7.1.3	Modifications to the Sprint Retrospective meeting	86
7.1.4	Waterfall process	87
7.1.5	Quality Gates	88
7.1.5.1	Definition of Ready	88
7.1.5.2	Definition of Done	88
7.1.5.3	Product Backlog Refinement	88

7.2	Secure Scrum	88
7.2.1	Requirement	91
7.2.2	Design	91
7.2.3	Implementation	91
7.2.4	Verification	91
7.3	Activities, tools and quality gates of this proposal	92
7.3.1	Requirement	92
7.3.1.1	Initiate Project Security Planning	92
7.3.1.2	Agile risk analysis: External Dependencies	93
7.3.1.3	Security Requirement Analysis	93
7.3.1.4	Agile risk analysis: Identify Trust Levels	94
7.3.1.5	Agile risk analysis: Identify Assets	94
7.3.1.6	Agile risk analysis: Entry Points	94
7.3.2	Design	94
7.3.2.1	Agile risk analysis: Identify Threats	95
7.3.2.2	Agile risk analysis: Ranking of Threats with DREAD	95
7.3.2.3	Agile risk analysis: Mitigation Strategies	96
7.3.3	Implementation	96
7.3.3.1	Document Security Controls	97
7.3.3.2	Static Code Analysis	97
7.3.3.3	Dynamic Code analysis	97
7.3.3.4	Pair Programming	97
7.3.3.5	Use of Dependency Checker	98
7.3.4	Verification	98
7.3.4.1	Pair Penetration Testing	98
7.3.4.2	Penetration testing	99
7.3.4.3	Code review	99
7.4	How this process handles the PAs of the SSE-CMM	100
8	Evaluation	103
8.1	Evaluation of agility, cost-effectiveness, and additional time-effort	103
8.1.1	Results of first subsection of our questionnaire	104
8.1.1.1	Results of questions regarding security activities and tools	104
8.1.1.2	Results of questions regarding security-related user stories	111
8.1.2	Discussion of the results	111
8.2	Evaluation of Secure Scrum in regard of security with OWASP-SAMM	111
8.2.1	Brief description of OWASP-SAMM	111
8.2.2	Evaluation with OWASP-SAMM	112
8.2.3	Results and discussion of results	112
9	Summary and Conclusion	121
9.1	Summary of methodology	121
9.1.1	Analysis of already existing security activities	122
9.1.2	Novel proposal for agile risk analysis (ARA)	122
9.1.3	Our secure Scrum process	123

10 Further work	125
10.1 Analysis of SE processes in regard of agility	125
10.2 Agile Risk Analysis (ARA)	125
10.3 Secure Scrum Process	126
A Evaluation of security activities of SE processes	127
A.1 Security activities of Microsoft SDL	127
A.2 Security activities of Cigital Touchpoint	136
A.3 Security activities of SREP	140
A.4 Security activities of NIST 800-64	145
Bibliography	153

List of Figures

3.1	The 12 agile principles behind the agile values.	15
3.2	Scrum skeleton, which presents the way of how software is developed in Scrum.	17
3.3	An example of a DFD for web application Threat Modeling.	23
3.4	Abuse case which illustrates a user who logs in to an application and a hacker who wants to login as a normal user.	25
3.5	Risk model of NIST SP 800-30.	26
3.6	Risk assessment process of NIST SP 800-30.	27
3.7	The three main areas of the SSE-CMM.	29
3.8	Process Areas of the Engineering Process of the SSE-CMM.	30
3.9	Risk assessment Process of the Engineering Process of the SSE-CMM.	31
3.10	Assurance Process of the Engineering Process of the SSE-CMM.	32
3.11	The NIST 800-64 secure system development lifecycle.	36
3.12	The Secure Development Lifecycle of Microsoft SDL.	37
3.13	The Security Requirements Engineering Process (SREP).	38
3.14	The seven Touchpoints for Software Security.	39
4.1	Example of a security-related user story of SAFECode.	44
5.1	Metric for evaluating the support of agile values (grades 0-2)	54
5.2	Metric for evaluating the support of agile values (grades 3-5).	55
5.3	Metric for defining how the speed of execution depends on iterative-/dividability.	55
5.4	Results of security activity analysis of Cigital Touchpoint in regard to how agile they are.	64
5.5	Results of security activity analysis of Microsoft SDL in regard to how agile they are.	64
5.6	Results of security activity analysis of NIST 800-64 in regard to how agile they are.	64
5.7	Results of security activity analysis of SREP in regard to how agile they are.	65
6.1	Our agile risk analysis.	75
6.2	NIST SP 800-60's metric for categorizing the impact of information systems on confidentiality, availability, and integrated.	77
6.3	Database scheme of our agile risk analysis method (iteration 1).	77
6.4	Example of our final database scheme (iteration 1).	78
6.5	Database scheme of our agile risk analysis method (iteration 2).	79
6.6	Example of our final database scheme (iteration 2).	79
6.7	Final database scheme of our agile risk analysis method.	81
6.8	Example of our final database scheme.	81

7.1	Waterfall structure of Secure Scrum process.	87
7.2	Components of the Secure Scrum Figure 7.3.	89
7.3	Secure Scrum process.	90
8.1	Evaluation results of each security activity of secure Scrum grouped by role.	106
8.2	Evaluation results of each security activity of secure Scrum grouped by security expertise.	107
8.3	Evaluation results of each security activity of secure Scrum in average.	108
8.4	Evaluation results of all security activities of secure Scrum in average grouped by role.	109
8.5	Evaluation results of all security activities of secure Scrum in average grouped by security expertise.	110
8.6	Security assessment of secure Scrum with OWASP SAMM.	113
8.7	Security assessment of secure Scrum with OWASP SAMM (threat assessment category).	114
8.8	Security assessment of secure Scrum with OWASP SAMM (security requirements category).	115
8.9	Security assessment of secure Scrum with OWASP SAMM (security architecture category).	116
8.10	Security assessment of secure Scrum with OWASP SAMM (design review category).	117
8.11	Security assessment of secure Scrum with OWASP SAMM (implementation review category).	118
8.12	Security assessment of secure Scrum with OWASP SAMM (security testing category).	119

List of Tables

4.1	How Scrum handles SSE-CMM's PAs	49
5.1	How Scrum handles SSE-CMM's PAs and which security activities analysed can be used to take a Process Area into account.	68
6.1	Agile risk analysis high-level description.	74
7.18	How Secure Scrum handles the Process Areas of the SSE-CMM.	101

Chapter 1

Introduction

1.1 Background

Agile software development is a software development process model for managing software projects [33]. It is an alternative to the traditional models. A famous non-iterative development model is the waterfall model and its variants, like the V-model [86]. Most of the existing security engineering processes are based on the waterfall model, and its variants [86], for example, Microsoft SDL [34], Digital Touchpoint [50], NIST 800-64 [47], and Common Criteria [45]. The waterfall model consists of the following phases [83]: requirement, design, implementation, verification and maintenance. These phases are performed linearly, steadily downwards like a waterfall. This structure poses many disadvantages, for example, one only knows if the software is working when the testing phase has been completed. This implies that the software is not shippable until the testing phase has been completed. Hence, the customer can only give feedback at the end of the process. If the software does not meet the requirements of the customer, much development time would have been wasted and changes would then be costly, because modifications in late stages of the process are more costly than in early stages. Moreover, the scheduling of a software project in a waterfall model is difficult, because the development team first has to implement the whole software before they can test their code and thus only know in late phases of the process if their implementation is actually working. Another problem is that if the money invested is already expended in the implementation- or the design phase, the software would not have been tested, which can lead to poor quality or even worse, no code might have been implemented at all. In contrast, agile development is based on short iteration cycles, early delivery and developing incremental software which allows users to quickly react to changes in business requirements [19]. Software increments pass through all waterfall phase in short iteration cycles. Thus, software increments can be delivered early and the customer can give quick feedback. If the software does not meet the requirements of the customer, the organization can react to this situation and change their further plans by adopting further software increments to fit the customer's needs. Nowadays, many companies have shifted to agile development processes, because of these advantages [101].

According to National Vulnerability Database (NVD) [67], currently many applications are vulnerable to cyber-attacks. Enterprise web applications are more vulnerable to such attacks than desktop applications, because open interfaces can be easily exploited once the software is deployed. Hence, the development of secure software and therefore the use of security engineering processes should be the main concern. An agile development process which is capable of efficiently producing secure web applications would be of interest.

1.2 Problem definition

Security requirements have to be defined and written down at the very beginning, because according to many authors, it is not possible to add security as an afterthought without immense extra effort and expense [26], [84], [31], [59]. Furthermore, to develop secure applications, threats have to be assessed, security requirements have to be defined, and a secure software architecture has to be created [21]. Next, one has to review the design, review the implemented code whether it contains security flaws, and finally implement assurance methods, like penetration testing.

The assessing of threats, defining of security requirements, and creating of a secure software architecture requires an considerable amount of planning-time and produces much documentation, which contradicts the principles of agile. In agile development, software is implemented in increments. Therefore, not all requirements are known beforehand and thus at the beginning of a project one cannot plan the whole software in regard to security. Agile development accepts the fact that requirements will often change and hence avoids time-intensive planning phases, because much planning time may be wasted if requirements change. Agile development also focuses on minimizing the effort for documenting and modelling. However, the creation of a secure software architecture requires modelling and the assessing of threats produces an considerable amount of documentation.

Moreover, pressure on time-to-market makes it difficult to take security measurements into account and to discover potential security flaws during the development of applications. Many automatic tools can help detecting security flaws but only to a certain degree. Agile development enforces the problem due to much shorter development iterations. As you can see it is a challenging task to combine the development of secure software with the principles of agile software development.

1.3 Intent of this Thesis

The intent of this thesis is to create a secure Scrum process, which fulfils all goals of a security reference model — the SSE-CMM. The System Security Engineering Capability Maturity Model (SSE-CMM) is an ISO/IEC standard (ISO/IEC 21827) [39] of a model that provides security engineering practices that an organization should implement to ensure the development of secure software. The SSE-CMM divides these practices into so-called process areas. Each process area consists of one or more goals and several security engineering practices (see Section 3.6 for greater details).

This secure Scrum process focuses only on the development of software and not its deployment or maintenance. We chose Scrum, because it is an increasingly popular agile development framework in practice. Security and agility is always a trade-off and depends on the level of security which should be achieved. The aim of this process is to achieve web application security to combat the most common application attacks regarding the OWASP Top 10 [77]. OWASP [72] is a well-known non-profit organization focused on web application security and its Top 10 project simply describes many common web application security flaws in practice (see Section 3.3 for greater details). In our opinion, trying to avoid these flaws is an effective first step for developing secure web applications.

1.3.1 What has Already Been Done in Literature

Though in literature several partial results exist to integrate security into agile processes [100, 96, 11, 82, 97, 46, 13, 57, 68, 10, 99, 18, 9, 29] (see Chapter 2 for a list of related papers) there is no consensus what the “best” and “most agile” way is and no complete secure agile process has been presented. Several authors suggested using security-related user stories or attacker stories in order to define security requirements and that security engineering activities should be added as Product Backlog items. Further highlights are that security-related user story templates should be used for identifying high risk components and Microsoft SDL’s Threat modeling method or brainstorming sessions would offer the possibility

of identifying vulnerabilities, threats and risks. Moreover, security experts should be involved in each team, static code analysis tools should be performed for security assurance and the "Definition of Done" should be extended in such a way that security engineering activities are mandatory.

1.3.2 Our contributions

Only partial results of an agile security process exist. The intent of this thesis is to create a whole agile security process. We suggest high level security activities which extend the Scrum development process and explain which Scrum roles are involved and which output of each activity is expected. In our opinion the following steps have to be performed to create an agile security process:

1. It has to be shown how security requirement engineering can be done in agile software development,
2. A method to analyse how agile security engineering practices and tools are has to be defined,
3. One has to incorporate agile-friendly security engineering activities and tools into an agile software development process to fulfil all goals of the SSE-CMM,
4. Vulnerabilities, threats and risks to an application have to be assessed in an effective and iterative manner, because software is implemented in increments,
5. Security testing has to be conducted in an efficient way in order to fit into agile.

This thesis mainly focuses on the last four steps. For the first problem we adopt already existing solutions and no new ones are introduced in this thesis. For the second point we adopt the method of [46]. Since the authors of [46] only loosely defined how the agility of security activities can be measured we refine this method in this thesis. For the third and fourth problem we identify the "most" agile activities and tools of already existing SE processes with the from us refined method of [46]. For the fourth problem, many authors [97], [57], [68], [97] suggest that Microsoft's Threat Modeling or sheer brain storming sessions should be used for identifying threats and risks to the application. However, in our opinion neither Microsoft's Threat Modeling nor sheer brain storming sessions should be used in agile. Microsoft's Threat Modeling method guides non-security experts to identify threats to the application in a prescriptive way. Nevertheless, Threat Modeling is a time-intensive method, because it requires creating several diagrams of the application, so-called data flow diagrams (DFDs), and it produces a considerable amount of documentation. In contrast, sheer brain storming about possible threats to the application without using methods or tools which support developers by this task does on the one hand not produce much documentation, and is more time-efficient than Microsoft's Threat Modeling approach. However, on the other hand to identify with sheer brain storming sessions as much threats as with Microsoft's approach, in our opinion, a person who actually knows many common attack patterns, and who is familiar with how software can be created securely is required. Such a person is not always available in practice. In this thesis we show an agile way of combining these two approaches. We combine the results of all steps to create a secure Scrum process. In the next Section our approach is explained in greater detail.

1.4 Our approach in further detail

Our approach is to integrate practices from non-agile secure development lifecycles into Scrum. Our approach consists of four phases. In the first phase we analyse in what way base practices of the SSE-CMM, which is a security reference model, can be integrated into Scrum. Secondly, we assess how agile activities and tools of well-established security engineer processes are and discuss how they can fit into an agile process. Next, we present an agile-friendly lightweight risk analysis method. The results of all these steps are then used to define a secure Scrum process. Finally, we evaluate this new process in order

to display how agile it is and how much security it provides. In the following sections we discuss the four phases in greater detail.

1.4.1 Integrating SSE-CMM practices into Scrum

The first step of our approach was to get an overview of what practices are needed with the help of the SSE-CMM to create a secure software lifecycle and to analyse how the base practices of this model can be integrated into Scrum. We searched in the definition of Scrum [87] for practices and activities which take any of the SSE-CMM practices into account and conducted a literature research to get ideas of how practices and activities of Scrum can be used to achieve goals of SSE-CMM process areas. We found out that Scrum lacks support for many process areas, especially for assessing security vulnerabilities, threats, their impacts and risks. It also does not describe any activity in order to specify security needs and to build assurance arguments. However, the coordination of security and the verification and validation of security can be provided by Scrum. Testing methods and code reviews which are recommended activities of Scrum [87] promote the verification and validation of security and thus, for example, the coordination of security can be accomplished with the following Scrum practices: small team communication, self-organized teams, Daily Standups, Product Backlog Refinement meetings, Sprint Plannings and Sprint Reviews. They can be used in the following way:

A Product Owner communicates security requirements to their teams and write them down formally in user stories and security related user stories as described in Section 4.1.3.2. Product backlog refinement meetings, which are ideally performed by more than one developer [14], aim at enumerating and documenting possible security risks, already defined security requirements and corresponding security constraints for specific user stories. Therefore, some kind of threat modeling or risk assessment can be used. In Sprint Plannings every issue is discussed with all other team members. Further, Product Owners highlight, prioritize and break down security issues. Developers pick the high prioritized security problems in daily standups. Because of the small team, self-organization and frequent face-to-face interactions security related matters will regularly be discussed with all team members.

1.4.2 Evaluation of activities and tools of existing SE-processes

As our second step we evaluated the agility of activities and tools of already existing security engineering processes in order to know which activities can fit into an agile development process. We evaluated security activities of the following well-known Security Engineering (SE) processes:

- NIST 800-64,
- Security Requirements Engineering Process (SREP) [52] which is based on Common Criteria,
- Cigital Touchpoint, and
- Microsoft SDL.

whether they can be used in the agile software development lifecycles. For this reason, we used the method to measure a so called agility degree described in [46]. This value describes the agility of an activity or tool and the level of an activity's or tool's compatibility with agile methodologies. If this value is high, the activity can be integrated into Scrum, and if not that means integrating it would be difficult, because one or more of the following *agile features* are not fulfilled [46]:

- simplicity,
- high customer interaction
- free of modeling and documentation,
- change tolerate,

- minimal speed of execution,
- people oriented,
- iterative / dividable and
- high flexibility.

Berkun [17] also states that flexibility, simplicity and change tolerance are features of agile methods.

1.4.2.1 Brief overview of the method and our modifications

Keramati and Mirian-Hosseiniabadi [46] used the agile manifesto [5] to extract these agile features and considered the *agile features* listed above as major characteristics of agile methodologies. Nevertheless, in our opinion the "high customer interaction" should not be considered, because many activities, for example, penetration testing, documentation of implemented security controls or static and dynamic code analysis, do not require customer interactions and would get zero points in the evaluation. This would not be fair from our point of view. High customer interaction is only important for defining security requirements or planning tasks, because the requirements should ideally be defined with the customer [8]. Mandatory high customer interaction would be a bottleneck in practice, because customers do not always have time for responses. Therefore, we removed this feature in our assessment.

To calculate the agility of an activity a grade in the interval [0, 5] is assigned to every agile feature. The agility degree of an activity is the sum of all measured feature grades. A low degree means that an activity is not "agile-friendly" and a high grade states that an activity is "agile-friendly". Since in [46] no metric has been defined to calculate these grades, we created a metric to make it clearer what a specific grade means. We state that if the speed of execution is very fast it is not necessary that it is iterative or dividable. Hence, we also added a second metric which defines the dependence of speed of execution on iterative-/ dividability. With these metrics we were able to calculate agility degrees for all the above mentioned SE processes.

For example, a security requirement analysis is no trivial task, because, in our opinion, a security expert is needed. In order to find security requirements, it does require a lot of documentation effort and planning, and allows medium tolerance for changes. Furthermore, it can be performed fast in the context of small user stories and is high informative, due to the fact that without such an analysis the security requirements are unknown. This activity is also highly people oriented, because it is not method based and requires an considerable amount of brainstorming, can be performed iteratively and flexibly, due to the fact that only the high important ones have to be identified in practice.

Regarding the fact that we determine the grades subjectively we decided to use a paper of Baca and Carlsson [13] to verify the results of our research. In their work, they have also analysed if activities of Microsoft SDL, Cigital Touchpoint and Common Criteria can be integrated into an agile software development lifecycle. However, they have chosen a different approach. Their results are based on a survey in which several employees of the company Ericsson AB were asked if specific activities of the above described SE processes can be used in their agile projects without losing too much agility.

1.4.2.2 Results

In accordance with our results, only about 25% of all from us analysed security activities are agile-friendly, such as the identification of security requirements, static- and dynamic code analyses, and manual code reviews. For 25% of the activities analysed we reached the conclusion that they are medium agile-friendly, because they fulfil all agile features only slightly more than to the extent of about 50%. An example for these activities is the defining of abuse cases, an attack surface reduction, and penetration testing. About half of the analysed activities do not fit into agile. Especially manual security testing-techniques and methods, and risk analyses.

1.4.3 Agile-friendly risk analysis

We found out that Microsoft's Threat Modeling approach [34, p. 101–130] and Cigital Touchpoints's, SREP's, and NIST 800-64's risk analysis approaches are not agile-friendly. However, these methodologies are important for the identification of threats, vulnerabilities, risks, their impacts and corresponding security controls. According to the SSE-CMM, the identification of these points is necessary in order to build secure software.

Instead of using these risk analysis/threat modeling methods, our approach is to create an agile-friendly risk analysis method. We therefore take OWASP's Application Threat Modeling (OATM) [79] as a reference and modify steps of it by integrating the following security activities: Cigital Touchpoint's "Abuse cases" activity, NIST 800-64's "Critical Assets"- and "Categorize Information System" activity, and Microsoft SDL's "Role Matrix"- and "Attack Surface Analysis/Reduction" activity. According to our results, these activities are medium agile-friendly. Abuse cases are used to identify threats and abuses to an application. The aim of the "Critical Assets" activity is to analyse what can be of value for an attacker, and the "Categorize Information System" activity is to prioritize the assets. The goal of the "Role Matrix" activity is to identify all user roles of the application and to determine their access restrictions to functionalities and assets of the application. An Attack Surface Analysis/Reduction is conducted to minimize the amount of possible threats to an application.

We have two highlights in our methodology. The first is that in order to identify threats in the software we adopt the use of so-called "security-related user story templates" [11] instead of decomposing the software with data flow diagrams (DFDs) as suggested by OATM and Microsoft's Threat Modeling approach. These security-related user story templates contain the following elements:

1. Security-related scenarios that have to be considered when developing applications, such as that a software architect has to ensure that all users have only access to resources which they are authorized to use,
2. It provides a list of security tasks which should be conducted in order to achieve the goal of a security-related scenario, and
3. each security-related user story template contains a link to a Common Weakness Enumeration (CWE) database entry [24] which is related to the security requirement of a story. This database enumerates several common weaknesses, and how one can prevent them as well as the most common failures.

We further combine the idea of abuse cases with these security-related stories. To each story we add several abuse cases with attack scenarios from the OWASP Top 10 in order to show several attacks which should be taken into account when a functional user story is implemented. This has the advantage, that the analysts are able to find threats by considering concrete attack scenarios, and to derive specific test cases from them. These test cases test whether a security-related user story is fulfilled.

The second highlight is that we introduce a database where a risk analyst performs this agile risk analysis methodology can quickly look which user is allowed to access a specific asset, sees the impact on it, and from which entry points an attacker can gain access to it. Furthermore, the risk analyst also sees which user story implemented the functionality that a user has access to a specific asset through a concrete entry point. This information should be used to support defining abuse cases. The main reason why we also add the user story where an asset or entry point was implemented is that the risk analyst sees which user story or security-related user story has to be revisited. These stories document the assumptions made, the attack scenarios considered, and further implementation details. In order to achieve a quick querying of this information we create a database scheme where we combine the listing of assets from OATM, the idea of rating their impacts from "Categorize Information System", and the idea of a "Role Matrix", where the access rights from all user to each asset are documented. Additionally, we also include the main concept of the "Attack Surface Analysis/Reduction" to identify entry points,

where an attacker can start attacks, such as html input forms. We state that this database is populated with data when the functionality of the user story defined is implemented which introduces new assets or which allows users to access an asset. In this way, this database is filled with data in an iterative way.

Moreover, also attack scenarios can be inserted into this database. In order to do so, first a security-related user story has to be inserted into the database. Secondly, through the access rights of a user on assets in this database can illustrate which access rights an attacker obtains in the course of a specific attack on a particular asset. Finally, the concrete abuse case of a security-related user story has to be attached to this scenario instead of a whole story.

1.4.4 Our evaluation

As a last step we evaluated the agility, cost effectiveness and time effort of our new agile Scrum proposal. In order to evaluate the secure Scrum process's agility, cost-effectiveness and roughly estimated additional time effort we conducted a survey in which we asked ten employees of a company which develops JAVA EE web applications with Scrum how agile friendly, cost-effective and time consuming the newly introduced practices, activities and tools are. We also performed an security assessment of the process with the help of the OWASP-SAMM evaluation framework [21]. This framework can be used to evaluate an organization's security process by calculating a maturity level which represents the provided security of software security practices.

1.4.4.1 Results

The results of the survey conducted shows that integrating security into an agile development process is not as cheap as we thought. The people who participated in the survey estimated that a user story which takes about 77h to plan, develop, and verify in common Scrum takes by using Secure Scrum about 108,2h (+45%). We asked for each activity, tool and practice in our process how agile it is by using 1 as very low agility or very low cost effectiveness and 5 as very high agility or very high cost effectiveness (1=very low, 2=low, 3=medium, 4=high, 5=very high). The participants hold the opinion that the process is medium agile and high cost-effective. The extensive search and categorization of threats, identification of assets, identification of trust levels and documentation of implemented security controls seemed to be too agile unfriendly and too cost-intensive. In our questionnaire we also asked in our questionnaire, whether security related user stories should be considered as normal items of the Product Backlog or whether an own security section should be added to functional user stories, which describes abuse cases of a normal user story. The results show that high important security requirements should be formalized as acceptance criteria in a security section of a functional user story and that stories of lower importance should be defined as security related user story and be linked to the corresponding functional story.

In our last step we performed a security evaluation of our new secure Scrum process by using the OWASP-SAMM evaluation framework for calculating a maturity level in different security aspects. We only performed an assessment of the construction and verification categories, due to the fact that our process only focus on the implementation of new functionality and the assurance that the implemented functionality is secure against the OWASP Top 10. According to our results, the process only lacks support for a secure architecture and design analysis.

1.5 Structure of the thesis and the methodology

The main purpose of Chapter 4 is to give the reader a general overview of which steps have to be taken in order to create a secure Scrum process by using the SSE-CMM. We argue how base practices of the SSE-CMM can be incorporated into Scrum and discuss on a high-level view which activities or tools can be used. They are explained in greater detail in the chapters below.

In Chapter 5 we evaluate whether activities and tools of existing SE-processes are agile-friendly. We use an already existing method in literature that uses the agile features of the agile manifesto in order to calculate an agility degree for each activity or tool. However, the authors of this method did not state how these grades of the features can be calculated, thus we introduce two metrics which can be used. After presenting the above mentioned method, this chapter presents our modifications to the method and our two newly introduced metrics. Finally, it illustrates our evaluation results of each activity and tool of Microsoft SDL, Cigital Touchpoint, SREP and NIST 800-64.

In Chapter 5 we found out that threat modeling and risk assessments are not agile friendly. Nevertheless, they have to be performed in some way in order to fulfil all goals of the SSE-CMM. For this reason, we introduce an agile friendly risk analysis method in Chapter 6. We modify OATM by removing steps from it that are from our point of view not agile friendly and integrate several agile friendly activities and tools of Chapter 5 for building a new agile friendly risk assessment. We further explain the database scheme mentioned in Section 1.4.3 with a simple example.

In Chapter 7 we describe a proposal of a secure Scrum process. However, this chapter does not describe a whole software development process in greater detail. It suggests high level security engineering activities and tools that extend the Scrum development process, which Scrum roles are involved and which output of each activity is expected.

Chapter 8 presents an evaluation of our secure Scrum proposal. First, we display the results of our survey in which we asked employee of a company, how agile friendly, cost-effective and time consuming each activity or tool of our Secure Scrum proposal is. This company develops Java EE Business to Business software and uses Scrum as a development lifecycle. Second, this chapter outlines the results of our OWASP-SAMM assessment. The outcomes of this assessment illustrate the level of security that this new secure Scrum provides.

Chapter 2

Related work

Several research papers have proposed approaches to add security to agile development. The first section of this chapter describes a paper that has focused on analysing whether Extreme Programming (XP) [8] is able to develop secure software. The second section outlines four papers that show how security requirements can be formulated and integrated into agile software development processes. The next section explains six papers that show how security engineering practices can be integrated into agile software development processes. The last section describes two papers that introduced agile risk assessment/threat modeling methodologies.

2.1 Analysis of Scrum

Our approach of analysing Scrum with SSE-CMM was inspired by Wäyrynen, Bodén, and Boström [100]. They analysed XP's suitability for implementing secure software and used the SSE-CMM as a check-list for evaluating which practices are already taken into account. They also showed some possible ways of how some SSE-CMM practices can be applied in XP. In the next paragraph we discuss some of their findings.

They found that XP does obviously not explicitly support any of the process areas of SSE-CMM except the verification of software and the coordination of security. On the one hand, software can be validated and verified with the help of the XP practices: test-driven development and following automatic tests. On the other hand, no security assurance arguments can be designed with XP, because of its lack of documentation. They state that XP lacks support for the assessment of vulnerabilities, risks, threats, their impacts and the specification of security needs. In advance security needs can be discussed with the customer, risks, threats and their impacts to the system must be found first, because without them the security controls needed are unknown. Security needs are often non-functional requirements, which cannot easily be broken down into estimable tasks, are not testable and are difficult to find. According to Wäyrynen, Bodén, and Boström [100], security needs should be defined with the customer, but the main focus of customers lies on functional requirements and not on non-functional ones. The finding of risks and the breaking down of non-functional requirements into estimable tasks is a non-trivial task, therefore, they suggest that a security expert should perform a risk assessment and then help customers to define security needs of a software system. They further suggest that the security expert should do pair programming with the development team and that static code analysis should be used to verify whether the software is secure.

We also think that some sort of security expert is needed for a risk assessment, because without security knowledge many threats of the attack surface can be uncharted. From our point of view, the analysts have to think like an attacker and therefore should have knowledge about many attacks, common vulnerabilities and weaknesses. For example, if the analysts do not know basic attacks, such as a "Man

In "The Middle Attack" or a "SQL Injection", in our opinion, they do not know how they can defend the software best. However, as stated by Wäyrynen, Bodén, and Boström [100] if only one in the team performs such a risk assessment only this person knows the threats. This should somehow be prevented.

We also want to integrate practices of the SSE-CMM into an agile development methodology, but in our case into Scrum. XP [8] and Scrum [87] develop software in increments and have a prioritized Product Backlog. Teams in both frameworks are self-organizing, which means that they plan on their own, how items in the Product Backlog are broken down into tasks, and when they tackle which tasks. In both cases these tasks are performed in time-boxed iterations, so called Sprints. However, in contrast to Scrum, XP teams have to follow many development practices, such as simple design, testing, refactoring, metaphor, collective ownership, coding standard and pair programming. These practices were very important in the work of Wäyrynen, Bodén, and Boström [100], because they explained how these practices can be used for security engineering base practices. Another difference between XP and Scrum is that Scrum has four mandatory meetings, such as Sprint Planning meetings, Sprint Reviews, Daily Scrum meetings and Product Backlog Refinement meetings. In Sprint Planning meetings the members discuss the tasks for the next Sprint. In Daily Scrum meetings and Sprint Reviews, backlog items can be changed, dropped or new prioritized, which keeps the process agile. By contrast, in XP, the planning and prioritizing of tasks is performed in the planning game. However, XP does not specify any practice to keep the process agile [30]. Through these differences we decided to perform another analysis, instead of completely relying on the results of [100].

2.2 Security requirements in agile development processes

Asthana et al. [11] suggests to use security-related user stories in order to identify security threats of functional user stories and presented a solution of how security requirements can be defined in Scrum. They defined about 36 general security-related user story templates, which describe non-functional security requirements and how they can be divided into small manageable tasks. These templates were created to support the formalization of security requirements in an agile development process.

We adopt the same approach and integrate it into our secure Scrum process.

Tuuli, Camillo, and Antti [96] also proposed a solution for defining security requirements in an agile project by using security-related user stories. A Product Owner should write specific security related user stories with the help of so-called security-related user story templates. These stories are then implemented by the development team just like ordinary functional ones. The authors displayed several templates that help creating stories for cloud-based software systems. The ISF Standard of Good Practice, secure software development lifecycle models, and ISO/IEC 27002 were used to define them. This approach is similar to [11]. The templates of [96] mainly focus on cloud-based software development, whereas Asthana et al. [11] defined their templates more generally and can therefore also be used for several software development domains. Thus, we decided us to adopt the approach of [11] instead of [96].

Pohl and Hof [82] suggested that security concerns of a functional user story should be formulated as security related user story, misuse story or abuse story. These security concerns should be linked to the corresponding functional user story in the Product Backlog with a so called S-Tag. Hence, security concerns of a functional user story are now Product Backlog items and are linked to functional user stories. The aim of such an S-Tag is to highlight backlog items that contain security critical tasks. They also suggest to extend the "Definition of Done" in such a way that verification of the security concerns is mandatory. If problems occur in verification they suggest that a new backlog item that only covers the verification part should be added to the Product Backlog.

However, in our opinion no additional backlog item should be added, because a user story is not completed before each part of it is tested. This would result in untested code, which can lead to unsafe code. Nevertheless, we adopt their approach to create backlog items for security concerns of a functional user

story and to link them in the Product Backlog. We formulate these security concerns in security-related user stories as described by Asthana et al. [11]. In this way we are able to formulate non-functional security requirements into security-related user stories, to break them into small manageable tasks and to highlight and to track them in the Product Backlog.

Vähä-Sipilä [97] presented how security risk management can be introduced into Scrum and what tasks should be performed by each role. Their main approach is to perform security requirement engineering with the help of generic "security story" templates and abuse cases and to use threat modeling to identify threats. In order to identify security requirements of user stories, the Product Owner should use generic templates, such as "As user my personal site should only be accessible by myself" to think about "positive" security requirements for specific functional requirements. If they want to define requirements that are not covered by the templates, they should use abuse cases instead, for example, "As an attacker I should not be able to bruteforce a user's password and get access to his personal side". Then, they break them down into estimable tasks and add them to the Product Backlog. Furthermore, the Scrum teams should use threat modeling of any kind in order to identify threats and risks that can occur if a functionality is implemented. To identifying which functional user stories require threat modeling, some kind of filter should be used, but did not state which filter can be used. Threat modeling and other security engineering tasks should be performed in Sprints and be added as backlog tasks in order to be visible to the Product Owner. In this way, the Product Owner can measure the costs more easily. In summary, Vähä-Sipilä [97] described a possible way of how security requirements can be defined in an agile software development process and how a threat analysis can be incorporated into such a process. Nevertheless, they did not describe which threat analysis method should be used.

We adopt their approach of formulating security-related user stories to specify security requirements and also integrate a threat analysis in an agile development process. However, in contrast to their work, in this thesis we describe an agile risk analysis in greater detail and further integrate several security activities into the agile software development process Scrum. The aim of this new process is to achieve web application security to combat the most common application attacks regarding the OWASP Top 10.

2.3 Agile security engineering activities

Keramati and Mirian-Hosseiniabadi [46] described a method in order to calculate how agile security engineering activities are. They used the agile values of the agile manifesto (see Section 3.1 for the definition) in order to measure the so called agility degree. This agility degree states how agile an activity is. We use this method to calculate the agility degree of several security engineering practices of well-established SE processes. Thus, we are able to integrate only the most agile activities in our secure Scrum process.

However, the authors of Keramati and Mirian-Hosseiniabadi [46] did not specify how these agile values can be measured. In this thesis, we present a possible metric.

Furthermore, Baca and Carlsson [13] conducted a survey at Ericsson, in which several employees of the company Ericsson AB were interviewed whether security activities of Microsoft SDL, Cigital Touchpoint and Common Criteria can be integrated into an agile development process. Each role in their development process has been represented by two or more employees. Questions from two perspectives were asked: how much would it cost to integrate such an activity and what benefit would it bring. At the end of their work, they presented a possible secure agile development process. We use their ranking of agile security engineering activities in order to verify our results. As mentioned above, we also want to know how agile each activity is.

Our approach is to use the method of Keramati and Mirian-Hosseiniabadi [46] combined with our own metric in order to calculate an agility degree. According to our results, if, an activity is just below our threshold to be agile, but their results showed that this activity is agile we say that it can be agile, if it is slightly adjusted in such a way that it supports one or more agile features better. If an activity is said

to be agile according to our parameters but not theirs, we discuss why they can be agile. In contrast to our work they only analysed whether activities or tools of the above mentioned SE process are agile and created without modifying them an high-level agile security process with them.

Microsoft [57] proposed how their Secure Development Lifecycle (SDL) process can be used in agile development processes by grouping their security activities and tools into three groups: every-Sprint practices, bucket practices and one-time practices. Every-Sprint practices are performed every Sprint, bucket practices do not have to be performed every Sprint, but frequently and one-time practices at the beginning of an agile project. However, they did not remove, add or modify any practice of the original SDL.

We adopt their approach and also divide the security activities of our secure Scrum process into how often they should be performed throughout the software development lifecycle.

Othmane et al. [68] introduced a high level agile software development process. They first extended the agile development process with high level security activities which they thought can fit in agile and second presented a methodology in order to reassure security of software increments with so called security assurance cases. The main idea of their first approach is to perform threat modeling or risk assessments and to use security-related user stories in order to develop secure software.

However, they never mentioned which methods should be used exactly and whether these methods are agile friendly. In our thesis we show that Microsoft's Threat Modeling, which is state of the art and risk assessment methods of Cigital Touchpoint, Common Criteria and NIST-800-64 are not agile friendly. Nevertheless, these kinds of methodologies should be used to identify threats, vulnerabilities and risks, because according to the SSE-CMM they are necessary to develop secure software. In this thesis we present a possible way of how risk assessments can be performed in a more agile way.

Antti [10] presented how security activities can be made visible in agile product management. They divided them into three main groups. The first are functional features, which represent features that can be tested with positive test cases, for example, the user should only be allowed to have five fail attempts for logging in. Engineering tasks, such as threat modeling or a web vulnerability scan, are the second type of security activities. The third group consists of "ways of working" activities. Tasks of this kind, such as ensuring the use of secure coding guidelines or security training of employees, are end-less tasks. The approach of the authors is to add all three types of activities to the Product Backlog. Security features should be formulated as features or attacker stories and be added to the Product Backlog. Developers formulate in later phases of the process attacker stories as specific features. "Ways of working" tasks should be written similar to a checklist. Each task is then defined as a specific security engineering task with a beginning and an end. Finally, security engineering tasks should be included as acceptance criteria to user stories or worked out as one-off tasks and pushed on the Product Backlog.

However, in contrast to our work, they did not describe a whole secure agile software development process, but rather grouped high-level security activities and how often they should be performed throughout a software development lifecycle and suggested to add all security activities to the Product Backlog. They did not describe these high-level security activities in greater detail.

Ville and Marko [99] proposed a solution on how high level security activities and tools can be integrated into Scrum. They first integrated, security and abuse cases, threat modelling/risk assessment, approving residual risks and flagging risky features into control points of Scrum, such as Sprint Planning and Sprint Review. Second, they evaluated with the in [46] described method if their solution is agile. The authors did not describe the above stated activities and tools in greater detail. The Product Owner should use security-related user story templates as stated by Vähä-Sipilä [97] and should approve residual risks in Sprint Reviews. According to their results, the first step is agile friendly, whereas the second is not. Threat modeling and flagging risky features should be performed in Sprint Plannings. Threat modeling is in accordance with their results not agile friendly, but necessary to build secure software and flagging risky features is agile friendly.

Just like in this thesis, Ville and Marko [99] analysed several security activities with the method described in [46] whether these activities fit into agile. However, they only analysed the following high-level

activities: threat assessment, high-risky feature flagging, residual risk approval, and security templates. In contrast, in this thesis we analyse 28 security activities from Cigital Touchpoint, SREP, Microsoft SDL, and NIST 800-64 whether they fit into agile.

2.4 Agile threat modeling/risk assessment

Antti [9] presented an agile risk assessment methodology. Risks are illustrated on a board in a 3x3 matrix format. The columns illustrate if the impact of a risks is low, medium or high and the rows if the probability is low, medium or high. A specific risk is calculated by multiplying the impact and the probability. For every risk a red sticky note is placed on the board. On this note a description of the risk, an identifier of the feature that is affected by this risk and the person who implements the feature are written. Yellow sticky notes represent possible mitigations for the risk and are attached to their corresponding red notes. In the planning phase, security training of employees is performed, security requirements and polices are defined, and attacker profiles are designed in order to plan against which attacker types the system should be protected. In the next step, two checklists are used to identify high-risk components. The first is based on past experiences made with high risk components, while the second one is founded on the idea of [97], where security and abuse cases are used. Then, in Sprint Plannings, risks are identified and assessed with the help of STRIDE, DREAD, TRIKE or brainstorming sessions and risk responses are created. A risk response can be, for example, "accept it", "reduce it" or "avoid it with security controls". After a response has been conducted, the related note on the board is crossed out and placed on another place on the board. In this way residual risks are clearly visible. In Sprint Reviews, the person who owned a risk approves it. If a residual risk is great, the risk owner has to explain to the Product Owner what the possible impact of this risk can be. Based on this explanation the Product Owner makes a decision on if this risk can be accepted. If for any reason that should not be the case, features that are affected by this risk are not shipped.

However, their approach is a high-level risk assessment and does not describe how threats can be found in greater detail. In this thesis we describe an agile-friendly method to identify threats in greater detail.

Ge et al. [29] illustrated a high level development process for developing secure web applications based on the Feature-Driven Development (FDD) and state that their approach fulfills all values of the agile manifesto. The main idea of the process is to create use-case and class/object models and to perform an iterative risk analysis using these models. In each development iteration a feature is implemented, use case or class/object models are updated and a risk analysis is performed. The results of the risk analysis of the previous iteration are used to perform the current risk analysis. In these diagrams assets are identified, which users are allowed to access an asset and how objects, classes and services interact with each other. With this knowledge attack paths are identified based on defined security policies.

Our approach is not based on using diagrams, because in our opinion the maintenance of such models is not cost effective. Instead we use security-related user story templates and steps of the OWASP Application Threat Modeling without using diagrams of any kind for our agile risk assessment proposal.

2.5 Summary

To integrate security into agile processes, several authors suggested using security-related user stories or attacker stories in order to define security requirements and that security engineering activities should be added as Product Backlog items. Further highlights are that security-related user story templates should be used for identifying high risk components and Microsoft SDL's Threat modeling method or brainstorming sessions would offer the possibility of identifying vulnerabilities, threats and risks. Moreover, security experts should be involved in each team, static code analysis tools should be performed for security assurance and the "Definition of Done" should be extended in such a way that security engineering activities are mandatory.

Though in literature several partial results exist to integrate security into agile processes, there is no consensus what the “best” and “most agile” way is and no complete secure agile process has been presented. The aim of this master thesis is to show a possible way of how to derive a Scrum-compatible secure software development process and to propose a whole agile security process.

Chapter 3

Preliminaries

3.1 Agile Manifesto

The manifesto for agile software development created and defined the term "Agile" [5]. It describes four values, which are supported by 12 principles. An example of such a value is that individuals and interactions are more important than processes and tools. Agile methods such as Scrum [88] and XP [8] approach these values. Alliance [5] defines the four *agile values* as follows:

“Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan”

Figure 3.1 shows the 12 principles.

1 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	7 Working software is the primary measure of progress.
2 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	8 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
3 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	9 Continuous <u>attention</u> to technical excellence and good design enhances agility.
4 Business people and developers must work together daily throughout the project.	10 Simplicity—the art of maximizing the amount of work not done—is essential.
5 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	11 The best architectures, requirements, and designs emerge from self-organizing teams.
6 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Figure 3.1: The 12 agile principles behind the agile values. Reprinted from [6].

3.2 Scrum

To understand further chapters of this work, we summarize Scrum in this Section according to its definition in Schwaber and Sutherland [88]. Scrum is an agile software development process which follows the agile manifesto. It is not a prescriptive process, but rather offers a framework and set of practices. Software is incrementally developed in so called Sprints. A Sprint is a period of time (30 consecutive calendar days or less) in which a software increment is implemented, tested, reviewed, and ready for shipping. Therefore, during a Sprint, all typical development phases of a waterfall process are performed. The Scrum team consists of three roles:

- Product owner,
- Scrum Master, and
- Development team.

Product Owner The Product Owner defines the vision of the project, its requirements, the return on investment (ROI) objectives and the release plan. In a so called Product Backlog all defined requirements are listed. These requirements can be functional and nonfunctional. This role is responsible for ensuring that all requirements are prioritized in this list in such a way that the ones with the highest value are developed first. Additionally, the purpose of this role is to clearly communicate the vision, requirements and priorities of the software to the development team.

Scrum Master The development team is kept concentrated on its Sprint goals by a Scrum Master. This role helps the team to follow the Scrum practices and teaches them Scrum. However, a Scrum Master has no authority over the development team and does not manage it. Another responsibility of this role is to provide a coordinator for the product owner and the development team.

Development Team The developers are grouped in small self-organising teams. A Development Team should ideally consist of between three to nine people. Their role is to implement the software. All members of a team are equal (cross-functional) which means that no strict roles such as testers or designers exist. Everyone within the team can help testing, designing, and implementing the software.

3.2.1 Definition of Done

All items of the Sprint Backlog have to be completed at the end of the Sprint. What “completed” means is defined in the so-called "Definition of Done". This definition is a list of requirements which must be met before a software increment is “completed”. This list varies from company to company and can, for example, include: the story has to be implemented, tested, reviewed and a documentation has to be written.

3.2.2 Scrum Events

Sprint Planning The Sprint Planning initiates a Sprint. It consists of two parts: Sprint Planning 1 & 2. Both last about four hours and they should not take longer than eight hours together. In the first the Product Owner talks with the team about the most important requirements. The team can ask the Product Owner what the purpose and intent of these requirements is and then selects requirements which they think they can turn into functionality in a single Sprint. In the second part of the Sprint Planning only the team takes part. They identify the tasks necessary to transform them into functionality and plan out how they actually achieve this. The resulting list of tasks is the so called Sprint Backlog. These tasks should form a potentially shippable product functionality.

Daily Scrum Every day in a Sprint, a Daily Scrum is held by the Development Team. It is strictly time-boxed to 15 minutes. In this meeting, developers discuss about what they have done the day before, what they are doing on this day and whether problems have occurred so that everybody involved is aware of the current status of the implementation. Furthermore, they can make adoptions in how to achieve the Sprint goal.

Product Backlog Refinement The Development Team decides on how and when a so called "Product Backlog Refinement" session is performed. The participants are Product Owner and the Development Team. The aim of this meeting is to create new backlog items, and to refine items which are already in the Product Backlog as detailed as possible in order to get them "ready". The Development Team can question the Product Owner in this meeting about the items in order to obtain a better understanding of the Product Backlog items. It should not take longer than 10% of the capacity of the Development Team.

Sprint- Review and Retrospective At the end of a Sprint, a Sprint Review meeting is held. The software increment developed in this Sprint is assessed according to the Sprint goals which are part of the Sprint Backlog. The participants are the Product Owner, Scrum Master, Development Team and any other stakeholder. The produced software increment is presented and ideally all planned items in the Sprint Backlog have been implemented. Items which have not obtained the "Definition of Done" are returned to the Product Backlog and re-prioritized. After the Sprint Review the Scrum Master holds a Sprint Retrospective meeting with his team. In this meeting they debate what went wrong in this sprint and also discuss possible ways of how they can improve their work in the next Sprint.

3.2.3 Scrum Skeleton

Figure 3.2 illustrates the skeleton of Scrum, which we adopted from [87]. First, high-prioritized items of the Product Backlog are selected from the Development Team and developed in iterations (Sprints). The result of Sprints are software increments. In 24-hour inspections (Daily Scrum) the whole Development Team meets and discusses if problems occurred during work, what goes according to plan, and whether adoptions should be made.

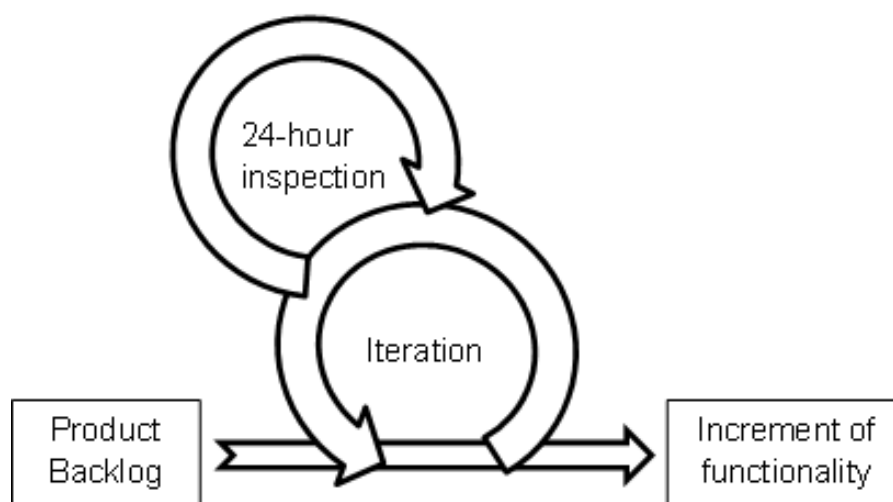


Figure 3.2: Scrum skeleton, which presents the way of how software is developed in Scrum. Reprinted from [87].

3.2.4 Further agile practices

The following agile practices are not part of the official Scrum defined by Schwaber and Sutherland [88], but are used in practice. We use them in our secure Scrum process.

3.2.4.1 User Stories

In this section we summarize the definition of user stories according to their definition in [4]. The creation of user stories is an agile requirement engineering practice. With the help of the customer or the Product Owner the Development team defines individual software increments in so called user stories. A user story consists of a one sentence brief summary of its business value and several acceptance criteria, which are written on a physical form, such as sticky notes. For defining a user story the INVEST [3] principle is applied, which is an acronym for:

1. **I**ndependent (it is independent of other user stories),
2. **N**egotiable (There is no specific contract for its outcome, but its details are rather added via collaboration),
3. **V**aluable (it provides a value to the overall product and the customer),
4. **E**stimable (it is defined in such great detail that a Development Team is capable of approximately estimate the effort to complete it),
5. **S**mall (it can be implemented in one Sprint), and
6. **T**estable (Acceptance criteria are defined in such great detail that test-cases can be derived from them).

3.2.4.2 Definition of Ready

A user story must fulfill the "Definition of Ready" [2] before it can be picked in the Sprint Planning to be implemented in a Sprint. This definition is a list of criteria which have to be met before a user story is immediately actionable.

3.2.4.3 Scrum Release Planning

According to [89] a Scrum Release Plan describes a plan for multiple Sprints. This plan is a list of requirements or user stories which should be shipped in a release. The plan can be date- or feature-driven. In the first case, functionality can be released frequently in a predefined time span, such as every three months. In the second case, the time span can vary from release to release depending on how long it takes to implement a certain functionality. For example, the implementation of a functionality A needs three Sprints whereas the development of functionality B requires about ten Sprints.

3.3 OWASP Top 10

The Open Web Application Security Project (OWASP) is an open community which was created to help organizations develop secure applications. This community started many projects to achieve this. One of its projects was to create a document which states the ten most critical web application security risks in practice — the OWASP Top 10 [77]. Furthermore, OWASP describes how these security risks can be prevented, provides several attack scenario examples and code snippets which illustrate security weaknesses which can be exploited by attackers in order to successfully launch an attack. The OWASP Top 10 project also display how likely it is for a threat agent to detect a weakness (easy, average, difficult)

within an application, and how often that is the case in practice (uncommon, common, widespread). Additionally, OWASP presents how difficult it is for one to exploit them (easy, average, difficult), the technical impact of such an attack (severe, moderate, and minor), and provides some attacker profiles. The OWASP Top 10 security risks:

- **A1 - Injection:** An attacker sends unescaped input to an interpreter. The interpreter is then tricked from the input to execute code or access data which the attacker normally has no access to.
- **A2 - Broken Authentication and Session Management:** Security flaws in the session management or the authentication allow an attacker to access unintended data.
- **A3 - Cross-Site Scripting (XSS)** First, unescaped input, such as JavaScript code is sent to a web application through, for example, a query. Second, this data is then somehow stored on the server. Finally, a user access the stored data via a web browser and the client's browser executes this JavaScript to steal information from the client. Another successful attack scenario is that the web application sends the untrusted and unescaped input directly to the client's browser without saving it.
- **A4 - Insecure Direct Object References:** If an internal object reference, such as a file, is exposed and no authentication is required for it and no other security controls are in place for it an attacker can access unintended data.
- **A5 - Security Misconfiguration:** If servers or frameworks are not configured securely, they are vulnerable to several cyber-attacks.
- **A6 - Sensitive Data Exposure:** If sensitive data is not protected with, for example, encryption an attacker may be able to steal or modify this data.
- **A7 - Missing Function Level Access Control:** The access rights of a function have to be checked every time when it is executed otherwise attacker might be able to forge requests.
- **A8 - Cross-Site Request Forgery (CSRF):** This attack requires a user who is logged on in an web application. An attacker forges a request and forces the logged on user to send the forged request to the web application that trusts the victim. In this way, the web application executes commands that the attacker wants it to be executed.
- **A9 - Using Components with Known Vulnerabilities:** By using known insecure libraries, frameworks, or functions an attacker might be able to exploit the known vulnerabilities within these components.
- **A10 - Unvalidated Redirects and Forwards:** Redirects and forwards to other sites have to be verified each time otherwise attacker might be able to lead clients to untrusted sites.

3.4 Preliminaries regarding risk assessment and threat modeling

In this section the generic approach of a quality risk assessment and threat modeling is described to obtain an basic understanding of both methodologies. To understand these methodologies we first explain the following key elements.

Asset

ISO/IEC TR 13335-1:1996 [41] defined *assets* as:

“anything that has value to the organization”

Threat ISO/IEC 21827:2008 [39] defined *assets* as:

“capabilities, intentions and attack methods of adversaries, or any circumstance or event, whether originating externally or internally, that has the potential to cause harm to information or to a program or system, or to cause these to harm others”

Threat agent ISO/IEC 21827:2008 [39] defined *threat agent* as:

“originator and/or initiator of deliberate or accidental man-made threats”

Vulnerability ISO/IEC TR 13335-1:1996 [41] defined *vulnerability* as:

“includes a weakness of an asset or group of assets which can be exploited by a threat”

Risk ISO/IEC TR 13335-1:1996 [41] defined *risk* as:

“potential that a given threat will exploit vulnerabilities of an asset or group of assets to cause loss or damage to the assets”

Residual risk ISO/IEC TR 13335-1:1996 [41] defined *residual risk* as:

“risk that remains after safeguards have been implemented”

Risk Model

A risk model defines risk factors, such as threat, impact, and vulnerability and describes how they relate to each other [64]. For example, that a threat exploits a vulnerability with a certain likelihood, which then causes an adverse impact.

Impact NIST [64] defined *impact* as:

“The level of impact from a threat event is the magnitude of harm that can be expected to result from the consequences of unauthorized disclosure of information, unauthorized modification of information, unauthorized destruction of information, or loss of information or information system availability.”

Threat source NIST [64] defined *threat source* as:

“A threat source is characterized as: (i) the intent and method targeted at the exploitation of a vulnerability; or (ii) a situation and method that may accidentally exploit a vulnerability. In general, types of threat sources include: (i) hostile cyber or physical attacks; (ii) human errors of omission or commission; (iii) structural failures of organization-controlled resources (e.g., hardware, software, environmental controls); and (iv) natural and man-made disasters, accidents, and failures beyond the control of the organization”

Threat event

A threat event is a sequence of actions, activities, or scenarios which exploits vulnerabilities [64]. They are initiated from threat sources.

Predisposing Condition NIST [64] defined *predisposing condition* as:

“A predisposing condition is a condition that exists within an organization, a mission or business process, enterprise architecture, information system, or environment of operation, which affects (i.e., increases or decreases) the likelihood that threat events, once initiated, result in adverse impacts to organizational operations and assets, individuals, other organizations, or the Nation. Predisposing conditions include, for example, the location of a facility in a hurricane- or flood-prone region (increasing the likelihood of exposure to hurricanes or floods) or a stand-alone information system with no external network connectivity (decreasing the likelihood of exposure to a network-based cyber attack)”

3.4.1 STRIDE

Microsoft created a threat categorization methodology, which is called STRIDE. In this section, we present a summary of STRIDE’s definition from Meier et al. [51, p. 16–17]. Its purpose is to categorize threats according to goals of attackers or particular exploits. STRIDE is an acronym for:

1. **Spoofing Identity:** Attackers should not be able to pretend to be someone other.
2. **Tampering with Data:** Attackers should not be able to alter stored data or data in transit.
3. **Repudiation:** It should be impossible for attackers to successfully deny that they performed illegal operations in a system.
4. **Information Disclosure:** It should be impossible for attackers to gain access to stored data or data in transit.
5. **Denial of Service:** Only authenticated and authorized users should be able to perform actions which require a lot of expensive resources like calculation effort or memory.
6. **Elevation of Privilege:** Attackers should not be allowed to elevate their privileges to perform actions they are not authorized.

STRIDE can directly be used for identifying threats from data flow diagrams (DFDs) with Microsoft’s STRIDE-per-interaction method [90]. A DFD is a model of an application that illustrates how data “flows” through an application. This method illustrates for each interaction between model elements of a DFD, which threat categories can occur. Microsoft provides with Microsoft Threat Modeling Tool 2016 [54] a tool in which DFD diagrams can be drawn and which automatically identifies threats with STRIDE-per-interaction.

3.4.2 Application Security Frame (ASF)

The Application Security Frame (ASF) [79] is a vulnerability categorization scheme. Thus, it is used to divide vulnerabilities of information system into groups. ASF uses the following vulnerability categories: authentication, authorization, configuration management, data protection in storage and transit, data validation / parameter validation, error handling and exception management, user and session management, and auditing and logging. It does not categorize threats from an attacker perspective, such as STRIDE, but rather from a defensive perspective.

3.4.3 DREAD

DREAD [51, p. 63–65] is a classification scheme for ranking threats. Each threat is ranked using five categories:

low=1, medium=2, high=3

Damage Potential	low = leaking trivial info	medium = sensitive	high = run as administrator
Reproducibility	low = very difficult	medium = requires timing window	high = very easy
Exploitability	low = extremely skilled person	medium = can be automated	high = amateur programmer
Affected Users	low = few users or anonymous users	medium = some users	high = all users
Discoverability	low = unlikely	medium = requires some thinking	high = very noticeable

Like STRIDE, it is also a method which was proposed by Microsoft and an acronym. In this case, DREAD is formed from the initial letters of each previously described category. In order to help team members to agree on risk ratings, this method is used to help calculate a risk value, which can be computed as follows:

risk value = (Damage Potential + Reproducibility + Exploitability + Affected Users + Discoverability)

As result, a value between 5 and 15 is obtained and represents the severity. A value between 12-15 is a serious risk, 8-11 a medium risk and 5-7 a low risk.

3.4.4 Threat Modeling

OWASP [79] defined *threat modeling* as:

“Threat modeling is an approach for analyzing the security of an application. It is a structured approach that enables you to identify, quantify, and address the security risks associated with an application. Threat modeling is not an approach of reviewing code, but it does complement the security code review process. The inclusion of threat modeling in the SDLC can help to ensure that applications are being developed with security built-in from the very beginning.”

3.4.5 Microsoft’s Threat Modeling method

Threat modeling is a risk analysis. In the definition of Microsoft’s SDL Threat Modeling consists of the following steps [34]: (1) define use scenarios, (2) gather a list of external dependencies, (3) define security assumptions, (4) create external security notes, (5) create one or more DFDs of the application being modelled, (6) determine threat types, (7) identify threats to the system, (8) determine risk, and (9) plan mitigations. Microsoft SDL adopted Threat Modeling from [94]. The goal of this method is to create a threat model for a specific use scenario. A threat model consists of a Data Flow Diagram (DFD) [94] model of the application and a list of threats which can occur in the specific use scenario.

In the first step, a specific use scenario of the application is defined. The next three steps are dedicated to the purpose of listing in which environment the applications is deployed and which security assumptions are made, such as relying on TLS to securely transport data over insecure connections. This information is documented in order that people who deploy and maintain the application are aware of these assumptions. They should be familiar with these details in order to be able to securely deploy and maintain the application.

In the next step from the specific use scenario of the application a DFD is modelled. In Microsoft’s Threat Modeling Tool 2016 [58] a DFD consists of the following elements: processes, external interactors, data stores, data flows, trust line boundaries, and trust border boundaries. Figure 3.3 shows such

a DFD diagram on hand of a web application. A square represents an external entity, whereas a single circle illustrates a process. A process is a software component that processes data, such as a web browser or a web application. External interactors, represent something or someone who interacts with a process, for example, a user or a third party software. Parallel lines are used to illustrate a data store, for example, a database or a file. An arrow displays in which direction the data flows and a dotted curve symbolize how the privilege levels change as the data flows. A dotted rectangle also represents a trust boundary. All elements in this circle is part of this trust boundary.

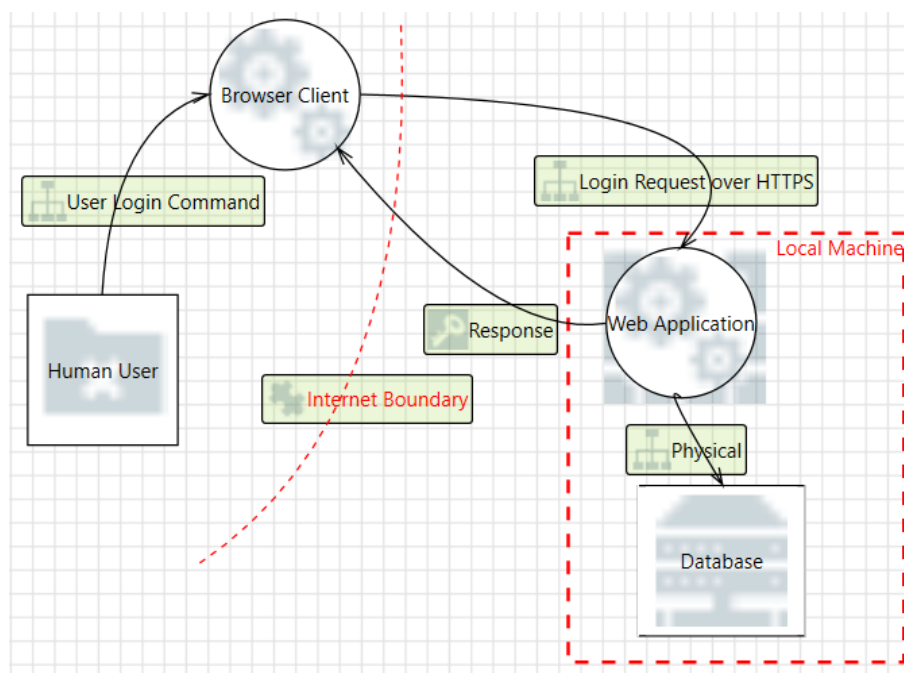


Figure 3.3: An example of a DFD for web application Threat Modeling. This picture was created by the author of this document with Microsoft's Threat Modeling Tool 2016.

In the sixth and seventh step with the help of STRIDE threats are determined from the DFD. As already mentioned in Section 3.4.1 Microsoft provides a tool which automatically generates threats from a DFD with the help of STRIDE-per-interaction, such as that the interaction between a process and a data store yields threats in the categories spoofing, and denial of service. The eighth step aims to prioritize the threats. For this purpose they suggest to use DREAD or a generic risk model with the following formula:

$$\text{Risk} = \text{Likelihood}(\text{probability that a certain attack is successful}) \times \text{Impact}(\text{damage of this attack}). \quad (3.1)$$

In the last step they suggest several options how risks can be handled. First, nothing can be done and therefore the risk is taken. The next option is to inform users about the risk or to transfer it via, for example, contractual agreements. The fourth possibility is to terminate the risk by removing the asset which the threat aims for. The last option is to mitigate it with countermeasures. For countering the threat with technology they provide a list which states for each STRIDE category several mitigation techniques, such as confidentiality and for each several mitigation technologies. For confidentiality, for example, they list ACLs and encryption.

3.4.6 OWASP Application Threat Modeling

In [79] OWASP specified a general threat modeling process which is very similar to Microsoft's Threat Modeling method. OWASP's method consists of three high-level steps. First, the application has to be decomposed, second threats have to be determined and ranked, and finally countermeasures and mitigations have to be determined. We summarize the steps from [79] below in more detail. The goal of this method is also to create a threat model for a specific use scenario.

3.4.6.1 Decompose the Application

In this step the reviewer should get familiar with the application, its assets, entry points (where an hacker can start an attack), trust levels (which external entities have access to which assets), and its interactions with external entities. Using this information DFDs, which illustrate how the application processes data is created. The subsections below describe which data types should be collected.

External Dependencies The aim of this step is to gather information about the dependencies of the application external to the code, such as on which system the application is deployed or which database is used. Only external entities which are important in regard to security should be documented. This information is written down in table format with an id of the external entity and a description of it as columns.

Assets We already defined them in Section 3.4. They are also documented in a table with four columns. The first column is dedicated to the purpose to present the id of an asset, whereas the second column states the name of it. In the third column a description of the asset is given and in the fourth the trust levels which have access to it are listed. More than one can be listed for an asset in a row. We will use the term "Assets Table" as abbreviation throughout the rest of this document.

Trust Levels Trust levels are external entities which have access to the application or which should not have access to it, such as hackers. They can also be web server user processes. These trust levels are listed in a table, which consists of three columns. The first displays the id of the trust level. The second illustrates the name of it, whereas the third contains a description of the trust level. We name this table "Trust-Levels Table" in this thesis.

Entry Points Entry points are interfaces where an attacker can submit data to the application in order to be able to start an attack, for example, a html input form, or a http port. According to the authors of [79] without such entry points no attack is possible. This information should also be documented in table format. The table consists of four columns. In the first an id presents the unique identifier of an entry point, whereas in the second column the name of it is illustrated. The third displays a description of it and the last column contains defined trust levels which are allowed to access it. In one row it is allowed that an infinite number of trust levels are listed. In this thesis we call this table "Entry-Points-Table".

Data Flow Diagrams The information gathered so far is used to design a Data flow Diagram (DFD). This is a modelling method to present this information more intuitively. DFDs are created exactly as stated by Microsoft's methodology which we already explained above.

3.4.6.2 Determine and Rank Threats

Determine Threats

The first step in order to determine and rank threats is to categorize them with, for example, ASF. ASF can support identifying threats from the defensive perspective whereas in STRIDE they can be determined from the attacker perspective. With the help of DFDs reviewers can identify potential targets of threats from the perspective of a hacker with the help of STRIDE. These targets can be, for example, data stores, or interactions with users. In order to defend them against attacks so called security controls have to be developed. ASF categorization can support the process of determining weaknesses of these safe guards by interpreting these weaknesses as threats. Furthermore, if available, a STRIDE or ASF categorized threat list with examples can be used to identify threats.

Abuse cases can then further be used to illustrate the vulnerabilities of security controls and show how they can be bypassed or even where security controls would be needed at all. Figure 3.4 shows how such an abuse case looks like. In this example, a use scenario is shown where an user enters his username and password to login to a web server and an abuse case where an hacker is illustrated who aims to attack the user authentication. Moreover, the use of threat trees is also suggested. According to the authors of [79], a threat tree is based on a tree structure, where the root node represent a threat goal. Nodes in the tree display vulnerabilities whereas leaves describe countermeasures.

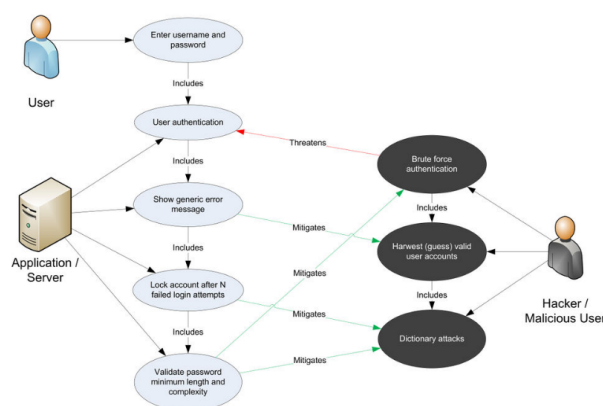


Figure 3.4: Abuse case which illustrates a user who logs in to an application and a hacker who wants to login as a normal user. Adopted from [79].

Rank Threats After threats are identified they are ranked. For ranking them OWASP suggests to use rating schemes, such as DREAD.

3.4.6.3 Countermeasure Identification and Mitigation Strategies

The focus of this step is to determine safe guards which can be used to prevent previous identified threats to exploit vulnerabilities. Due to all threats should have been identified with STRIDE or ASF so far they suggest that a ASF & Countermeasure List or a STRIDE Threat & Mitigation Techniques List is used for determining security controls. These lists provide for each category of STRIDE or ASF which high-level countermeasures should be applied. For example, in the ASF Threat & Mitigation Techniques List is illustrated that for preventing threats in the authorization threat category strong ACLs and role based access controls should be used. They state that threats can be either fully mitigated, partially mitigated or not mitigated. Additionally, they describe the same five ways of how threats can be handled in risk management as in Microsoft's method: do nothing, inform about the risk, mitigate it, accept it, transfer it, or terminate it.

3.4.7 Risk assessment

According to ISO/IEC 27005:2011 [40], the aim of a risk assessment is to determine assets, and to identify vulnerabilities and threats which can cause an impact on this asset. Moreover, it focuses on determining security controls which reduce the resulting risk of vulnerabilities and threats, and it ranks them. In the next subsection we show an example of an risk assessment methodology in practice.

3.4.7.1 NIST SP 800-30 Guide for Conducting Risk Assessments

The authors of NIST SP 800-30 [64] describe a general risk assessment and how it should be conducted. Figure 3.5 shows their general risk model which is used throughout this whole methodology. It states that a threat source initiates with a specific likelihood a particular threat event(s), which further exploits with a certain likelihood a vulnerability with a specific severity. This vulnerability is exploited in the context of predisposing conditions with a certain pervasiveness. Furthermore, for determining the likelihood of the successful exploitation of the threat event the effectiveness of planned or already implemented security controls are also considered. This exploitation causes with a certain degree an adverse impact. The combination of the overall likelihood and the impact produces then a organizational risk.

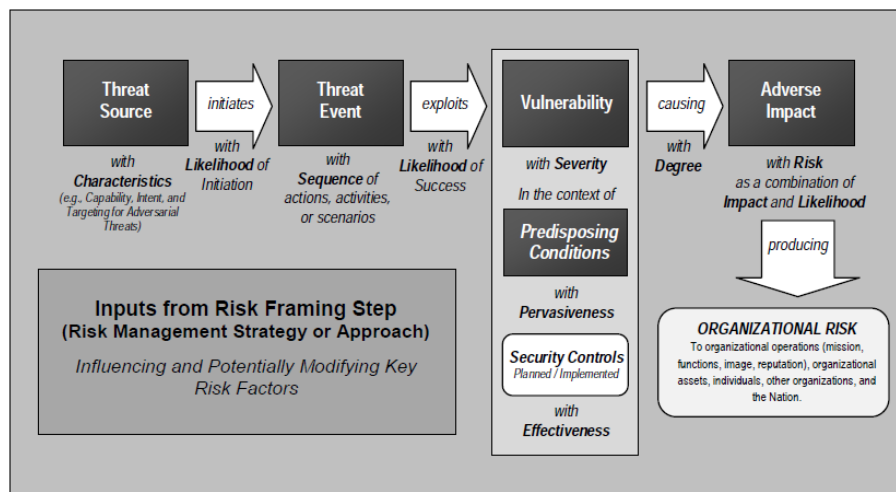


Figure 3.5: Risk model of NIST SP 800-30. Adopted from [64].

They categorize the assessment into three tiers. In the first risk is identified from the organizational level, which focuses on, for example, management activities whereas the second is the mission/business process level that concentrates, for example, on the enterprise architecture. The last tier aims to evaluate risk on the information system level, such as applications, networks, and information system components. Figure 3.6 shows of which steps this methodology consists. We summarize each step from [64] below:

Prepare for the assessment In the first step the purpose of the assessment is identified in regard of what the specific outcome should be. As second point, the scope of the analysis (hardware, software, system interfaces, user, system mission, and data and information) is identified and also in which timeframes it should be examined. The aim of the next step is to determine on which assumptions and constraints the assessment should be based on. It consists of points, such as defining risk tolerances, impacts of attacks, and which threat sources and vulnerabilities are considered. Next, the source inputs of vulnerabilities, threats and impacts are identified, for example from security reports. NIST provides exemplary tables which list where they can be taken from. These tables list possible source inputs for each tier. Finally, the risk model and assessment- and analysis approaches are chosen. For the risk assessment type they can choose whether it should be based on nonnumerical categories or levels

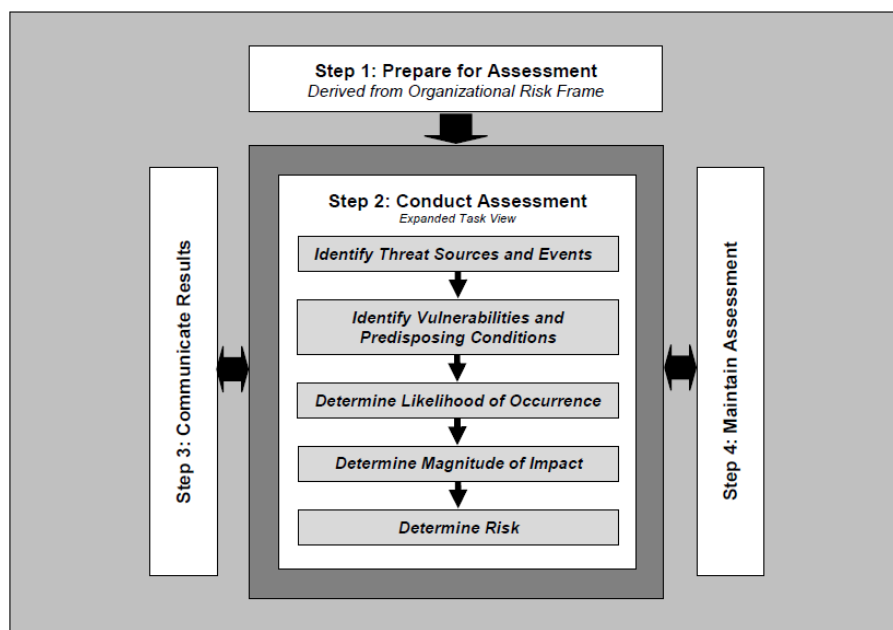


Figure 3.6: Risk assessment process of NIST SP 800-30. Adopted from [64].

(qualitative) or based on the use of numbers (quantitative) and for the risk analysis approach they can select if it should be threat-oriented, asset/impact-oriented, or vulnerability-oriented. The only difference between these risk analysis approaches is on whether the analysis starts with identifying threats, or attacks on assets and the resulting adversary impact, or on determining vulnerabilities. Starting from another order can bias the results.

Identify Threat Sources and Events In this point first threat sources are determined and as second step threat events which can be initialized from the identified threat sources are analysed. They provide several tables to support this process. For example, they supply a table which lists several types of threat sources, such as outsider of the system, trusted insider, fire, hurricane, or an operating system. They further provide tables which state scales of how adversary- capabilities, intents, and targets can be assessed with numeric vales from 0-100 and also with grades from very low to very high.

As next step threat events, such as that an adversary inflects the information system by sending malware via email are identified. Threat events caused from non-adversary are also considered, such as that an earthquake destroys servers. NIST provides tables which list several exemplary threat events with a detailed description.

Identify Vulnerabilities and Predisposing Conditions This step aims at analyzing vulnerabilities and rating them by severity with either a numerical value or with grades from very low - very high. Next, predisposing conditions are determined and rated by pervasiveness. Pervasiveness states the degree on which tiers it applies, for example, whether it applies to only few tier(s) or to all of them.

Determine Likelihood of Occurrence In this point the overall likelihood of that a threat occurs is identified. First, the likelihood of that an adversary initiates a threat event is evaluated, or the likelihood that non-adversary threats occur. As third step the likelihood that a threat event actually results in an adverse impact is rated. Finally, the overall likelihood is calculated by combining these likelihoods. NIST provides again exemplary tables of exemplary assessment scales.

Determine Magnitude of Impact Based on the threat events, vulnerabilities and predisposing conditions analysed in the previous steps and the effectiveness of planned or already implemented security controls the level of impact is calculated. The impact of a threat is evaluated, for example, whether threats harm individuals, assets or operations.

Determine Risk Finally, the risk is calculated as a combination of the overall likelihood of a threat and its impact. For example, if the likelihood is very low, but the impact is high the resulting risk would be medium.

3.5 Common Criteria

Common Criteria is a security assurance standard, the ISO/IEC 15408 standard. It consists of three parts [36, 37, 38]. Its aim is to assure that a system is specified, implemented, and evaluated in an accurate way. To evaluate whether the developed information system is secure Common Criteria states that first the Target Of Evaluation (TOE) has to be defined. A TOE represents the product or system which is verified. Second, in a Protection Profile (PP) common user security requirements are determined by a group of users (for example consumers). Third, a Security Target (ST) document is specified, which contains security objectives of the TOE, a description of the TOE and the environment in which it is deployed. Furthermore, it contains Security Functional Requirements (SFRs), Security Assurance Requirements (SARs), IT security requirements, and measures to meet these requirements. A SFR can, for example, describe a class of requirements that a TOE has to meet in order that this TOE is capable of protecting user data. SARs provide measures to ensure that SFRs are fulfilled, such as testing and performing vulnerability assessments. CC provides a set of several SFRs and SARs. In so called Evaluation Assurance Levels (EALs) CC specifies the depth of assurance requirements. These EAL's specify to which extent an evaluation is performed. They range from one to seven. The first states that the product is only tested functionally whereas to obtain EAL 7 the product has to be formally verified. CC states that obtaining an EAL above EAL 4 is not economically feasible for many companies. Through comparison of evaluated STs with PPs customers are capable of identifying whether their requirements are fulfilled by the product. One ST can meet requirements of several PPs. CC enforces developers to create documentations in order to provide evidence that the software actually meet user requirements of a PP and SFRs in a ST document.

3.6 System Security Engineering Capability Maturity Model

The System Security Engineering Capability Maturity Model (SSE-CMM) is an ISO/IEC standard (ISO/IEC 21827:2008) [39] of a model that provides security engineering base practices. These base practices describe security engineering practices that an organization should implement to ensure the development of secure software. SSE-CMM is used to introduce security engineering practices into a software development lifecycle (SDLC) and to evaluate the capability level of such practices in a SDLC. SSE-CMM defines the following list of security-related process areas: (1) Administer Security Controls, (2) Assess Impact, (3) Assess Security Risk, (4) Assess Threat, (5) Assess Vulnerability, (6) Build Assurance Argument, (7) Coordinate Security, (8) Monitor Security Posture, (9) Provide Security Input, (10) Specify Security Needs, (11) Verify and Validate Security.

Each process area consists of a number of base practices and one or more goals. The process areas of SSE-CMM can be divided into three main areas: (1) Engineering Process, (2) Assurance Process, (3) Risk Process. Figure 3.7 illustrates how these processes work together.

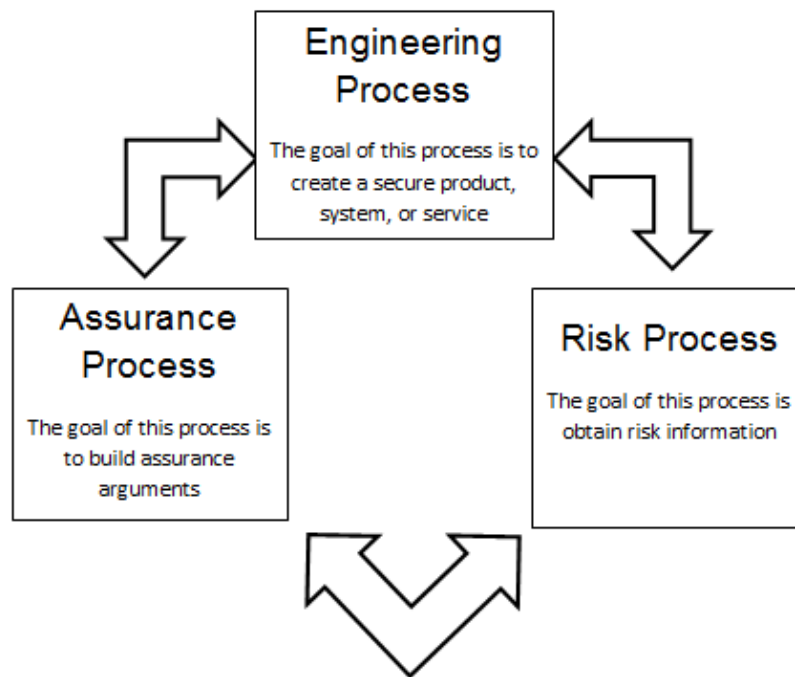


Figure 3.7: The three main areas of the SSE-CMM. Reprinted from [39].

First, in the risk process risk information are identified. Second, in the engineering- and assurance process the identified risks are taken into account and together they ensure that the software is developed in a secure way and mitigates these risks. Finally, the assurance process validates whether all claimed security- objectives and requirements are met. The engineering process focuses on configuring security controls, coordinating security, tracking of incidents, and on monitoring and maintaining the application in regard of security. Additionally, in these process security requirements, policies, and standards are specified, and possible security implications are identified. The goal is that everyone is aware of them and can make considerations how to tackle them. The aim of the risk process is the identification of impacts, risks, and vulnerabilities of an application. The assurance process aims on identifying security assurance objectives, and on verifying and validating whether all claimed security- objectives, and requirements are met. Figure 3.8 lists all process areas of the engineering process, their goals and base practices. Figure 3.9 lists all process areas of the risk process whereas Figure 3.9 illustrates all process areas of the assurance process. We adopted the process areas and their goals and base practices from [39].

Engineering Process

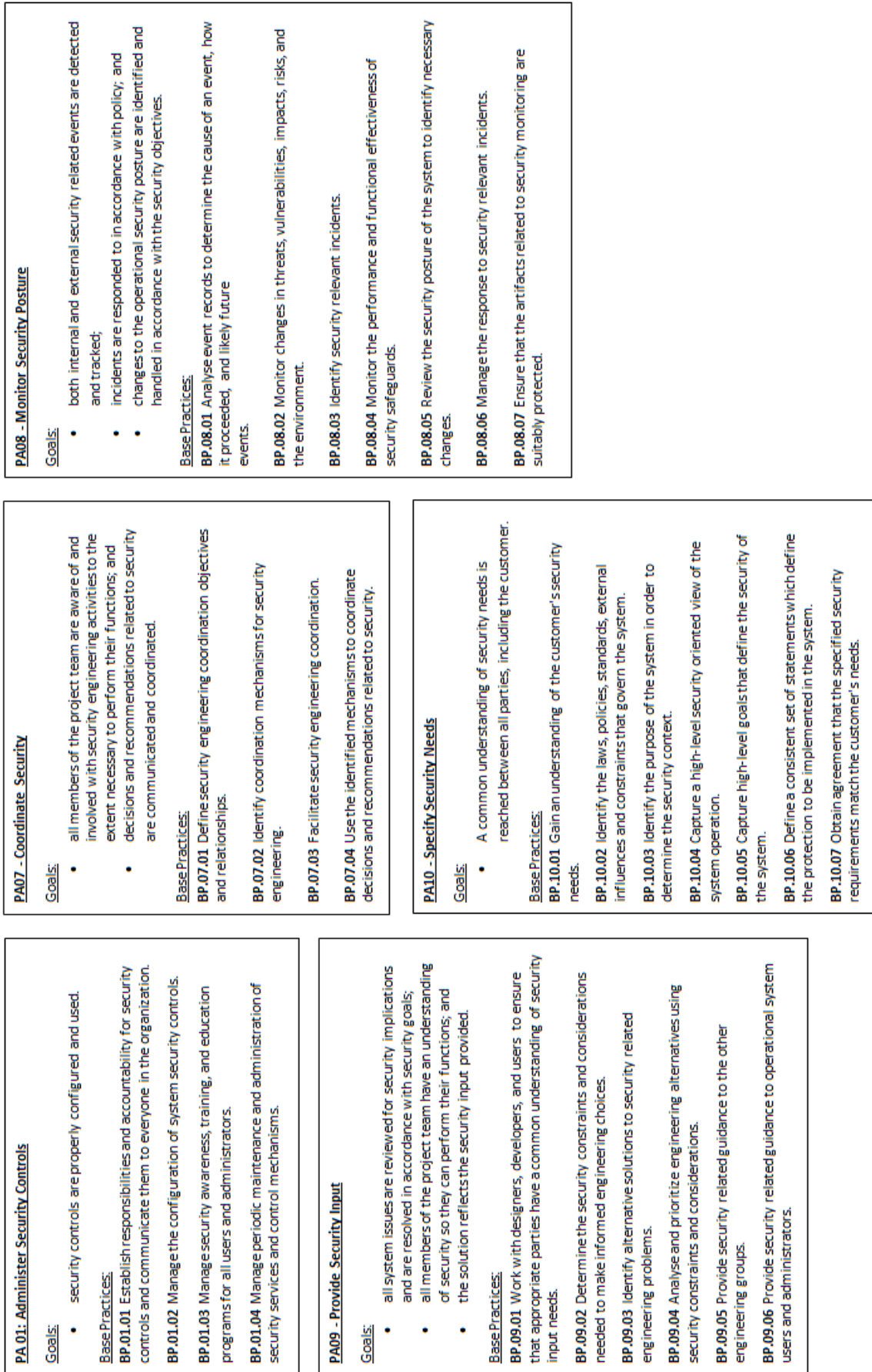


Figure 3.8: Process Areas of the Engineering Process of the SSE-CMM. Reprinted from [39].

Risk Process

<p>PA 02: Assess Impact</p> <p><u>Goals:</u></p> <ul style="list-style-type: none"> the security impacts of risks to the system are identified and characterized. <p><u>Base Practices:</u></p> <p>BP.02.01 Identify, analyse, and prioritize operational, business, or mission capabilities leveraged by the system.</p> <p>BP.02.02 Identify and characterize the system assets that support the key operational capabilities or the security objectives of the system.</p> <p>BP.02.03 Select the impact metric to be used for this assessment,</p> <p>BP.02.04 Identify the relationship between the selected measurements for this assessment and metric conversion factors if required,</p> <p>BP.02.05 Identify and characterize impacts.</p> <p>BP.02.06 Monitor ongoing changes in the impacts.</p>	<p>PA 03: Assess Security Risk</p> <p><u>Goals:</u></p> <ul style="list-style-type: none"> an understanding of the security risk associated with operating the system within a defined environment is achieved; and risks are prioritized according to a defined methodology. <p><u>Base Practices:</u></p> <p>BP.03.01 Select the methods, techniques, and criteria by which security risks for the system in a defined environment are identified, analysed, evaluated, and compared.</p> <p>BP.03.02 Identify threat/vulnerability/impact triples (exposures),</p> <p>BP.03.03 Assess the risk associated with the occurrence of an exposure.</p> <p>BP.03.04 Assess the total uncertainty associated with the risk for the exposure.</p> <p>BP.03.05 Order risks by priority.</p> <p>BP.03.06 Monitor ongoing changes in the risk spectrum and changes to their</p>
<p>PA 05 - Assess Vulnerability</p> <p><u>Goals:</u></p> <ul style="list-style-type: none"> an understanding of system security vulnerabilities within a defined environment is achieved. <p><u>Base Practices:</u></p> <p>BP.05.01 Select the methods, techniques, and criteria by which security system vulnerabilities in a defined environment are identified and characterized.</p> <p>BP.05.02 Identify system security vulnerabilities.</p> <p>BP.05.03 Gather data related to the properties of the vulnerabilities.</p> <p>BP.05.04 Assess capability and motivation of threat agent for threats arising from man-made sources.</p> <p>BP.05.05 Assess the system vulnerability and aggregate vulnerabilities that result from specific vulnerabilities and combinations of specific vulnerabilities.</p> <p>BP.05.06 Monitor ongoing changes in the applicable vulnerabilities and changes to their characteristics.</p>	<p>PA 04 - Assess Threat</p> <p><u>Goals:</u></p> <ul style="list-style-type: none"> threats to the security of the system are identified and characterized. <p><u>Base Practices:</u></p> <p>BP.04.01 Identify applicable threats arising from a natural source.</p> <p>BP.04.02 Identify applicable threats arising from man-made sources, either accidental or deliberate.</p> <p>BP.04.03 Identify appropriate units of measure, and applicable ranges, in a specified environment.</p> <p>BP.04.04 Assess capability and motivation of threat agent for threats arising from man-made sources.</p> <p>BP.04.05 Assess the likelihood of an occurrence of a threat event.</p> <p>BP.04.06 Monitor ongoing changes in the threat spectrum and changes to their characteristics.</p>

Figure 3.9: Risk assessment Process of the Engineering Process of the SSE-CMM. Reprinted from [39].

Assurance Process

<p><u>PA06 - Build Assurance Argument</u></p> <p><u>Goals:</u></p> <ul style="list-style-type: none"> the work products and processes clearly provide the evidence that the customer's security needs have been met. <p><u>Base Practices:</u></p> <p>BP.06.01 Identify the security assurance objectives.</p> <p>BP.06.02 Define a security assurance strategy to address all assurance objectives.</p> <p>BP.06.03 Define measures to monitor security assurance objectives.</p> <p>BP.06.04 Identify and control security assurance evidence.</p> <p>BP.06.05 Perform analysis of security assurance evidence.</p> <p>BP.06.06 Provide a security assurance argument that demonstrates the customer's security needs are met.</p>	<p><u>PA11 - Verify and Validate Security</u></p> <p><u>Goals:</u></p> <ul style="list-style-type: none"> solutions meet security requirements; and solutions meet the customer's operational security needs. <p><u>Base Practices:</u></p> <p>BP.11.01 Identify the solution to be verified and validated.</p> <p>BP.11.02 Define the approach and level of rigour for verifying and validating each solution.</p> <p>BP.11.03 Verify that the solution implements the requirements associated with the higher level of abstraction.</p> <p>BP.11.04 Validate the solution by showing that it satisfies the needs associated with the previous level of abstraction, ultimately meeting the customer's operational security needs.</p> <p>BP.11.05 Capture the verification and validation results for the other engineering groups.</p>
--	---

Figure 3.10: Assurance Process of the Engineering Process of the SSE-CMM. Reprinted from [39].

3.7 Preliminaries regarding software testing

In this section we give some definitions of different testing techniques. First, we explain some general testing techniques and then we focus on security testing methods, tools and techniques.

- In "unit testing" [43, p. 77], developers test small units of code separately, which are no larger than a class.
- Integration testing [43, p. 229–244] tests the interaction between interfaces or units in the low-level design.
- System testing [43, p. 253] tests the system end-to-end in a production-like environment in order to make sure that the business functions defined in the high-level design are working correctly .
- Black box [15] testing means that a tester tests a software system by attacking a system from outside and has no knowledge about the target system and source code. For example, blackbox testing can be integration and system testing but not unit testing [60]. For unit testing the code has to be known.
- Contrary to black box testing, in whitebox testing [60] the code is completely known by the tester. The tester use this knowledge to design test cases. For example, whitebox testing can be unit and integration testing, but not system testing [60], because system testing emulates the behaviour from outsiders such as customers.
- In grey box testing the targeted software is partially known and is a combination of whitebox testing and blackbox testing [20].

3.7.1 Security Testing

OWASP [78, p. 27] defined *security testing* as:

“A security test is a method of evaluating the security of a computer system or network by methodically validating and verifying the effectiveness of application security controls”

Security testing can either be white-box, grey-box, or black-box testing [18]. There are several special types of it which can be used in practice. We list several of them that are discussed in this thesis, but we do not provide a full list of methods, tools and techniques that can be used.

3.7.2 Black-box web vulnerability scanner

Black-box web vulnerability scanners [15] are automated testing tools that use common web attacks, such as SQL Injection (SQLI) and Cross-Site Scripting (XSS) to attack a running system in order to gain knowledge about its vulnerabilities. They simulate an attacker that does not know the source code of the target system. At first, webpages of the web application are crawled, then malicious inputs that can trigger a vulnerability are automatically entered, and finally the behaviour of the application is observed [27]. There are a number of open source and commercial tools available, such as Acunetix Web Vulnerability Scanner [1] and IBM AppScan [35].

3.7.3 Penetration testing

The aim of this methodology is to attack the software in the same way as a hacker might, namely by identifying vulnerabilities and then exploiting them. According to Ma, Kupzog, and Paul [49], it can be used in three variants: blackbox, whitebox, and greybox and can be performed manually by so called pentester or automatically with the help of tools. By the manual approach, a pentester usually gathers as

much information as possible of the target system, identifies its vulnerabilities and exploits them. Automatic "all-in-one" penetration testing tools do not require a pentester, such as Metasploit [53]. However, there is another possibility, for example, web vulnerability scanners such as OWASP ZAP [74] can be used to first automatically find vulnerabilities and then to help a pentester to manually exploit them.

3.7.4 Static code analysis

Static code analysis tools [23] read source code or compiled code and use some sort of pattern searching based on "rules" in order to identify security flaws without actually executing code. They can be included into compilers and are used to detect common coding failures, such as buffer overflows and out-of-scope memory. Furthermore, web application attacks, such as, SQL Injection can also be tested with these tools [75]. Many tools first construct some model of the source code and then use control flow analysis and data-flow analysis for identifying where input values may affect other values in the code at some point of interest [48]. In this way, only statements or functions which are called, because a certain value has been submitted from an outside user are taken into account. Furthermore, only variables which have been affected by the submitted value are considered. On the one hand, if a so called malicious input is, for example, not escaped until it reaches an unsafe method, a vulnerability is detected by the tool. On the other hand, if the input value has been validated before an unsafe method has been entered, no vulnerability is detected. Thus, only functions which can be potentially called, because a user has entered something, are detected as vulnerabilities. As result, the rate of false positives is minimized. However, according to [48] these tools are not sound and therefore are not able to deliver any false positives.

3.7.5 Dynamic code analysis

Dynamic code analysis [42] helps to identify subtle coding errors and examines a program in a real or virtual environment. It is therefore a runtime verification tool and hence differs from static code analysis. It uses either introspective code or platform emulation in order to understand the program's current runtime state. Many tools of this kind, such as Valgrind [98] Microsoft Application Verifier [55] are, for example, able to detect uninitialized memory, dynamic memory leaks, or race conditions and can be included into compilers. According to [32], this kind of tool is more precise when it comes to identifying vulnerabilities in the source code and is also better suited for run-time programming languages, such as threads and polymorphism than static code analysis tools.

3.7.6 Red team testing

Red Team Testing [81] is a testing method where security specialists that have the expertise and resources of hackers (read team) perform cyber-attacks in a real-life scenario on a system or network to test whether it is vulnerable. Additionally, they also test whether the employees are capable of detecting such an intrusion and how long it takes them to recognize it and if they are able to defend against it. The testing usually includes penetration testing of some system or network components. The attacks can also include social engineering, where, for example, phishing mails are sent to employees in order to break into the system. The team is typically hired from the customer and gets the permission to attack the system or network. Beforehand, the read team defines with the customer which attacks are performed. It is also important to mention that the red team must document each successful and unsuccessful attack. If for any reason the attack is not successful, the customer has proof that the system is not vulnerable to such an attack. By contrast, in the case of a successful attack, the attack can be reproduced with the help of the documentation.

3.7.7 Fuzz Testing

Fuzz testing or fuzzing [69] is a software testing technique that is based on the black box principle. A so called fuzzer tool automatically executes test cases and aims to cause a system crash, which states that the software contains vulnerabilities. This tool executes test cases by submitting input values or files, for example, in forms of web applications. In each test case it either randomly alters input, or mutates valid input, or generates input that makes sense to the program. Due to the black-box approach, these tools cannot locate vulnerabilities in the code. Furthermore, one can test with the help of these tools for a particular set of vulnerabilities whether they can be exploited.

3.8 Security Engineering processes

In this section we give a brief summary of the Software Engineering (SE) processes NIST 800-64, Microsoft SDL, SREP, and Cigital Touchpoint and describe each of their security activities. In Chapter 5 we analyse the activities of them in regard to their agility and describe the activities in greater detail.

3.8.1 NIST 800-64

Kissel et al. [47] defined a secure system development lifecycle — NIST 800-64. As all of the processes which we describe in this Section it is based on the waterfall model. It consists of the following phases: initiation, development and acquisition, implementation and assessment, operations and maintenance, and disposal. Each phase consists of a set of security activities and control gates. The aim of the control gates is to ensure that all security activities of a phase are sufficiently accomplished before a new phase is started. Figure 3.11 illustrates all of its security activities and control gates and also states in which phase each is located.

Initiation In the initiation phase first security milestones, and the security vision of the information system are planned in the "Initiate Security Planning" activity. The goal is that all stake holders and people involved in creating the information system are aware of these points. Second, in the "Categorize the Information System" activity assets of the information system and their impact on availability, confidentiality, and integrity are assessed. The "Assess Business Impact" activity aims to identify business processes and to evaluate their impact on availability, confidentiality, and integrity. Furthermore, resources which are needed to recover critical systems are identified, backup strategies are determined, the costs of unavailability are assessed, and roles and responsibilities for these tasks are identified. The next phase conducts a privacy impact assessment in which components where sensitive information is processed, or stored is assessed and the severity on loss is conducted. The last activity of this phase "Ensure Use of Secure Information System Development Processes" focuses on planning and performing security training sessions for the development team, and to develop and follow coding standards. Furthermore, it aims to establish a proper quality management, to create and use secure coding patterns, and to use authentication for code repositories.

Development/Acquisition The aim of the development/acquisition phase is to conduct a risk assessment (Assess Risk to System activity), to select and document proper security controls for the threats identified in the risk assessment (Select and Document Security Controls activity), to design a secure architecture of the application (Design Security Architecture activity), and to properly document these steps (Develop Security Documentation activity). Moreover, in this phase functional- and security testing is performed (Conduct testing activity).

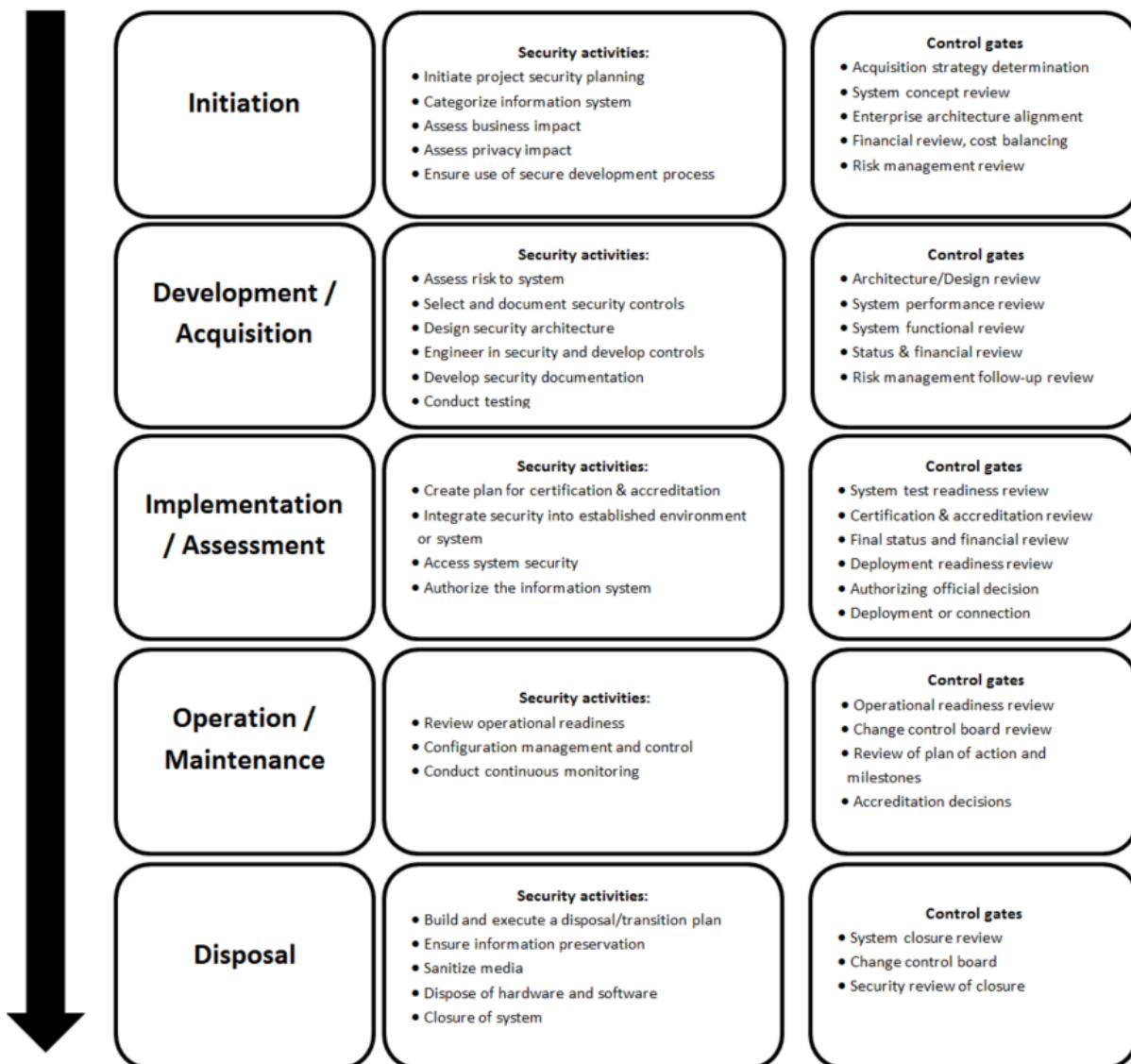


Figure 3.11: The NIST 800-64 secure system development lifecycle. Reprinted from [47].

Implementation/Assessment First, in the implementation/assessment phase the activity "Create a Detailed Plan for C&A" is performed. In this activity a plan is created which states how security-certification and accreditation is performed. The involved persons are determined, the scope of testing, and it is discussed which data are needed to make a decision with respect of which residual risks are acceptable. Second, in the "Integrate Security into Established Environments or Systems" activity the application is deployed on an operational environment. Additionally, in this activity all security controls are enabled and acceptance- and integration testing is performed. Next, in the "Assess System Security" activity a security certification is conducted in which the focus lies on verifying whether all security- and privacy requirements are met by the current specified software design, and selected security controls. Furthermore, the effectiveness of the implemented security controls is periodically tested. Finally, in the "Authorize the Information System" activity a security accreditation is performed in which based on the effectiveness of the implemented security controls and the acceptable residual risk a decision is made whether an information system is authorized to process, store, or transmit information.

Operation/Maintenance The operation/maintenance phase consists of three activities. The aim of the first (Review Operational Readiness) is to modify tests of security controls if significant changes occur to the system. The purpose of the "Perform Configuration Management and Control" activity is to configure the software, hardware, and firmware of the system, to manage changes on for example, the hardware of the system, and to determine the security impact of them. The last activity of this phase "Conduct Continuous Monitoring" focuses on controlling whether the implemented security controls remain effective over time.

Disposal The initial activity of the last phase disposal, is to create and execute an disposal/transition plan. This plan focuses on which steps have to be performed when a information system is closed down, transitioned, or migrated and what has to be done with its stored information. Furthermore, information preservation is ensured, media are sanitized, and the hardware and software is disposed securely. For example, storage mediums were private information was saved are destroyed. Finally, the system is actually shut down.

The control gates of each phase have the purpose to review the above listed activities whether they were performed properly. The first three phases also include financial reviews.

3.8.2 Microsoft Secure Development Lifecycle (Microsoft SDL)

Microsoft SDL [34] is a software development security assurance process. It states several security activities that can be incorporated into all phases of a software development lifecycle. It consists of seven phases: training, requirements, design, implementation, verification, release, and response. Figure 3.12 illustrates all of them and states which security activities are located in which phase.

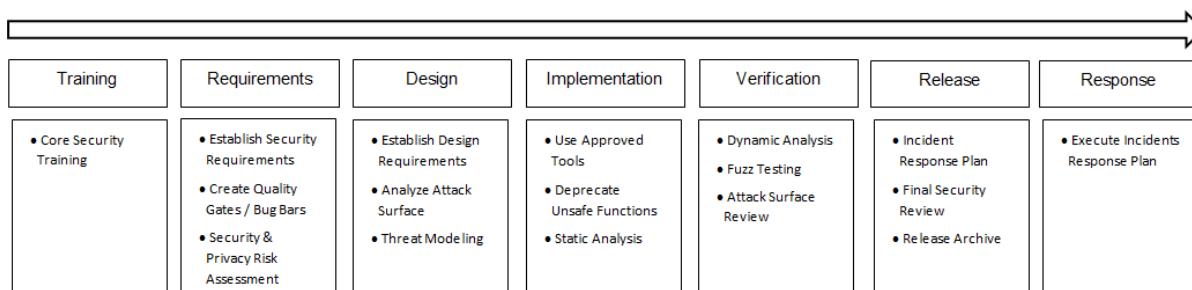


Figure 3.12: The Secure Development Lifecycle of Microsoft SDL. Reprinted from [56].

Training The first phase of the SDL "Training" consists of only one activity "Core Security Training". This activity ensures that employees are trained in effectively applying all practices of this process.

Requirement In the requirement phase SDL states that first security- and privacy requirements are defined. Next, quality gates and bug bars are created. They define the criteria for meeting a minimum acceptable level of security- and privacy quality. The last activity of this phase performs a security- and privacy risk assessment. In the security risk assessment the parts of the project which require activities, such as threat modeling, and penetration testing are identified. In the privacy assessment is analysed which components of the applications they process, store or transmit information, and to what degree.

Design The third phase of the SDL "Design" focuses on the creation of a secure software design, to perform an attack surface analysis/reduction, and to apply threat modeling. For creating a secure design, design requirements are established. The aim of an attack surface analysis/reduction is to minimize

the opportunity that attacker can exploit vulnerabilities within the application through, for example, layered defenses, and authorization. In threat modeling the focus lies on identifying assets, threats, and vulnerabilities in a structured way (we already described their threat modeling in greater detail in Section 3.4.5).

Implementation The implementation phase focuses on implementing the software in a secure way. In order to achieve this, SDL states that static code analysis tools should be used, approved security tools should be established, and old and unsafe APIs should be deprecated.

Verification To verify that the application is secure, in the verification phase dynamic program analysis and fuzz testing tools are used to test the application at run-time. Furthermore, threat models and the attack surface are re-reviewed to ensure that they take changes in the design and implementation phases into account.

Release In the release phase, an incident response plan is created, a final security review conducted, and the software is certified and data which is needed to maintain the software is archived. An incident response plan defines what should be done when an incident, for example, a cyber attack occurs, and when threats, vulnerabilities or when bugs are found after the software is released. Furthermore, emergency contacts which should be contacted in incidents are defined. In an final security review all artefacts which were produced throughout applying security activities are reviewed again. In an security certification is examined whether all security requirements are met.

Response In the last phase response the incident response plan is actually being performed on a incident or when vulnerabilities emerge.

3.8.3 Security Requirements Engineering Process (SREP)

Security Requirements Engineering Process (SREP) [52] is a process based on Common Criteria (we explained Common Criteria and its components in Section 3.5), which describes activities that aid the integration of security requirements in a software development lifecycle. Furthermore, this process uses the SSE-CMM and assurance requirements of Common Criteria to evaluate the provided security of an information system. Figure 3.13 lists all of its activities and how they integrate Common Criteria in their process.

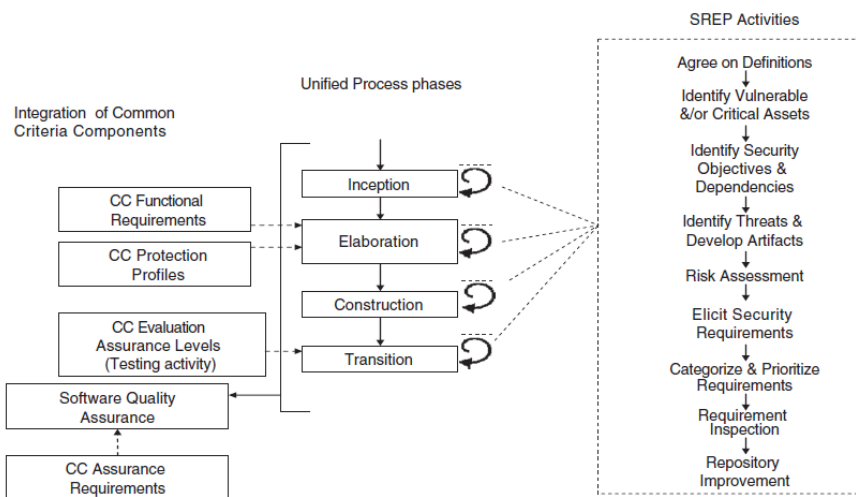


Figure 3.13: The Security Requirements Engineering Process (SREP). Adopted from [52].

In the "Agree on Definitions" activity the customer and the organization agree on the security vision of the project and the security policies. The next one aims to identify assets which are vulnerable to attacks and for which it would be critical if they were abused. Further, the "Identify Security Objectives & Dependencies" activity focuses on identifying security objectives and dependencies from, for example, legal requirements, critical assets, likely attacker types, and defined security policies. In the fourth activity artefacts, such as abuse cases, or attack trees are created to derive further threats from them. Next, some sort of risk assessment is performed. SREP does not state which exact methodology should be used. In the next activity "Elicit Security Requirements" security requirements are derived from the previous performed risk assessment. Next, already determined requirements are categorized and prioritized. The last two steps are dedicated to the purpose to perform a requirement inspection and a repository improvement. In the requirement inspection the determined security requirements are validated whether they are complete and verifiable. Furthermore, all artefacts produced so far are reviewed and the SSE-CMM, Common Criteria's assurance requirements and EALs are used to evaluate the security of the software. In the repository improvement model elements and threats which are likely to occur again in further developments are documented.

3.8.4 Cigital Touchpoint

McGraw [50] defined seven touchpoints (set of best practices) that can be applied in software development lifecycles in order to achieve the development of secure software. We call them Cigital Touchpoint throughout this thesis. These touchpoints are: security requirements, abuse cases, architectural risk analysis, risk-based security tests, code review, penetration testing, and security operations. Figure 3.14 illustrates when these touchpoints should be performed in a software development lifecycle. Throughout the development of software several artefacts are produced. This Figure outlines to which artefacts which touchpoints should be applied.

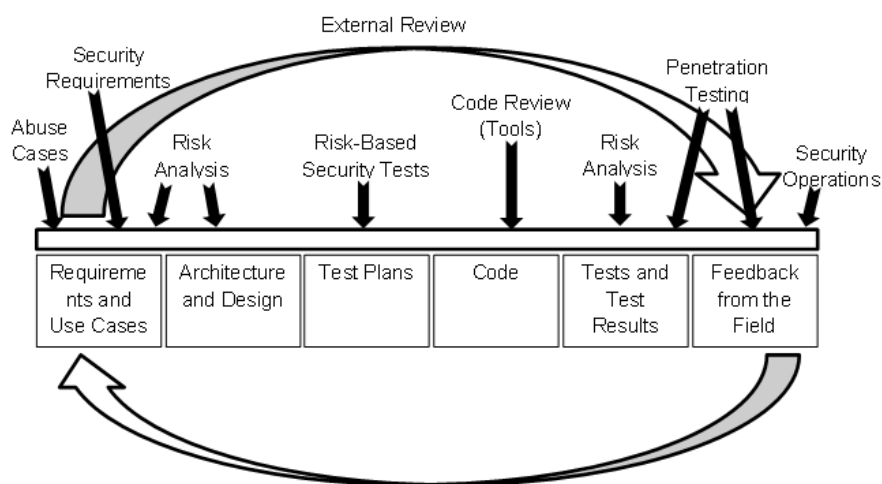


Figure 3.14: The seven Touchpoints for Software Security. Reprinted from [50].

Requirements and Use Cases First, Cigital Touchpoint states that additional to defining requirements and use cases also security requirements and abuse cases are defined. Security requirements are derived from, use cases, laws and regulations, commercial considerations, contractual obligations, and from results of risk analyses.

Further security requirements are defined through abuse cases, which describe uses cases when the application is abused. These scenarios state how security controls can be bypassed and which illustrate where security controls are missing at all.

Architecture and Design In the architecture and design phase a risk analysis helps to identify design flaws in regard of security.

Test Plans Complementary to ordinary functional test plans also test plans which put their focus on testing the security of the software are created. They test whether the implemented security controls are working as intended. These tests can consist, for example, of unit and integration tests. Furthermore, tests which simulate attacks of hackers are developed. All tests are prioritized based on the risks obtained from risk analyses. Thus, security tests which test high-risky features are executed first.

Code To evaluate whether the code produced is actually secure code reviews with static code analysis tools are conducted.

Tests and Test Results Next, the planned risk-based security tests are actually executed. Additionally, red team- and penetration testing is performed (we describe these two testing techniques in greater detail in Section 3.7).

Feedback from the Field In the last touchpoint "security operations" developers should share their security expertise with security operation employees. The work of security operation employees consist of deploying the software in an environment, on monitoring it, and to perform, for example, penetration testing. Developers should come together with security operation employees in order to help them to deploy the software securely, and also to support them on security incidents.

Chapter 4

Analysis of Scrum

In this Chapter we define which security practices (SSE-CMM *security engineering base practices*) need to be integrated into Scrum in order to develop “secure” web applications and we check whether Scrum supports some of SSE-CMM’s *security engineering base practices*. If for any reason Scrum does not support a SSE-CMM *security engineering base practice* we discuss how it can be integrated into Scrum. For example, we discuss the necessity to integrate a risk analysis/threat modeling method in Scrum. However, to integrate several particular *security engineering base practices* into Scrum it is necessary to incorporate specific security- activities, methods, techniques, or tools. Therefore, we evaluate in Chapter 5 which concrete security- activities, methods, techniques, and tools of well-established SE processes can fit into agile.

As discussed in Section 3.6, SSE-CMM is an ISO/IEC standard of a model that consists of several process areas (PAs). These PAs consist of several *security engineering base practices* and one or more goals. A security engineering *security engineering base practice* describes a high-level security engineering practice that an organization should implement to ensure the development of secure software.

4.1 SSE-CMM’s Processes

The PAs of the SSE-CMM are divided into three main groups: risk, engineering and assurance. In the following sub-sections we will list all groups and their process areas, briefly summarize all goals and *security engineering base practices* of each PA and evaluate which *security engineering base practices* can be supported by Scrum. A description of the SSE-CMM and a detailed listing of all *security engineering base practices* of each process area can be seen in Section 3.6. The PAs are listed in the order in which they should be performed according to SSE-CMM. The focus on PA01 and PA08 lies on project management, staffing and organisational practices, which are not specific security engineering tasks. Thus, we did not consider these process areas in the sub-sections below.

4.1.1 Risk Process

Summary of SSE-CMM’s risk process

The risk process includes PA 02 ("Assess Impact"), PA 03 ("Assess Security Risk"), PA 04 ("Assess Threat"), and PA 05 ("Assess Vulnerability"). The main focus in this process is the assessment of threats, vulnerabilities, their impacts and the resulting risks. All of these steps are related to each other and can be combined in a threat modeling or a risk assessment (both methodologies are described in Section 3.4).

Discussion of how PAs of the SSE-CMM's risk process can be integrated into Scrum

A possible way to fulfil the goals of the SSE-CMM risk process' PAs is to use threat modeling or a risk assessment [44]. All *security engineering base practices* of PA two, four and five except the monitoring of vulnerabilities, threats and their impacts are addressed by threat modeling. A risk assessment also includes these steps and additionally rates and prioritizes risks. Both methods are applied in various ways in practice. For example, Microsoft's Threat Modeling, Cigital Touchpoint's risk analysis and TRIKE [85] have applied them in different ways. Nevertheless, a considerable amount of time is required to perform any of them in the planning and design phases, because all possible threats, vulnerabilities and assets of a piece of software have to be analysed and ranked. After these tasks have been performed, the analysts should have knowledge about all possible vulnerabilities that can be exploited by an attacker and how much damage each attack would cause. From our point of view, security expertise and domain specific knowledge is needed in order to complete these tasks without leaving a great amount of the attack surface uncharted. In the opinion of several authors, threat modeling is seen as too difficult to implement and too costly to integrate it into an agile process [13], [12]. According to our results in Chapter 5, where we analyse security activities and tools of several well-established SE processes, we came to the same conclusion. We think that to perform such heavy weight methods for every user story is too much planning overhead for an agile process. In order to reduce the planning effort there are in our opinion two naive ways.

Different options to add *security engineering base practices* of PAs of SSE-CMM's risk engineering process into Scrum

1. The first possibility would be to integrate it into the "Definition of Ready". This definition can be extended in such a way that performing this methodology is only mandatory for medium or high risk features.
2. Another possibility would be to add it as a Product Backlog item for each medium or high-risk user story.

In both cases, that would require some method to identify security risky components. However, identifying risky components is the intent of a risk analysis/threat modeling method. Beforehand, only the obvious high security risky components are known. Therefore, to perform risk analysis/threat modeling methods for high security risky components only is no option to reduce the effort of these methodologies in an agile process.

Proposal of how the *security engineering base practices* of SSE-CMM's risk process can be integrated into Scrum

Due to the fact that existing risk analysis- and threat modeling methods are not agile friendly, and because performing such methods for only high risky components is no option we create in Chapter 6 a new risk analysis method that fits better into agile.

4.1.2 Engineering Process

Summary of the SSE-CMM's engineering process

The engineering process focuses on providing security input (PA 09), the specification of security needs (PA 10) and the coordination of security (PA 07). Security needs describe what security controls need to be implemented in order to prevent risks to the system. Hence, possible risks to the system have to be identified before security needs can be defined, which is covered by the above described risk process. PA 09 defines some *security engineering base practices* in order to achieve the goal that all people in the development process have knowledge about all possible risks and how they can be mitigated. Hence, it

is not enough if one person in the team has knowledge about all possible security information. PA 07 focuses on the coordination of security, how information has to be communicated between the different parties and that all parties are involved in the security engineering process.

4.1.2.1 PA 09 ("Provide Security Input") and PA 10 ("Specify Security Needs")

Discussion of how PA 09 ("Provide Security Input") and PA 10 ("Specify Security Needs") can be integrated into Scrum

In this section we discuss how the goals of PA 09 and PA 10 can be achieved in Scrum. Before user stories are implemented in Scrum the planning and high level design is performed [30]. In this phase functional security requirements, such as: "As an user I am only allowed to login five times", can be added as acceptance criteria in user stories, but non-functional requirements are difficult to add, because they cannot easily be broken down into estimable tasks and are not simply testable [100]. In our opinion, security requirements have to be defined and written down in user stories at the very beginning, because according to many authors, it is not possible to add security as an afterthought without immense extra effort and expense [26], [84], [31], [59]. Before a user story is in Scrum implemented, Product Backlog Refinement meetings are performed where user stories are analysed by one or more developers [87]. The goal of it is to find a way to implement the requirements specified in a user story. If the developer or developers are already aware of common weaknesses and have some security related knowledge, we suggest that they discuss in the Product Backlog Refinement meeting about possible security threats. In order to identify threats they can perform something like a risk assessment. Nevertheless, this implies that many developers have some prior security expertise, which can often not be the case in practice. An obvious solution would be that all developers attend a security training, but we think that there are two other options in order to solve this problem.

Different options to add *security engineering base practices* of PA 09 ("Provide Security Input") and PA 10 ("Specify Security Needs") into Scrum

The first option is to add a security expert to each Scrum team. His role would be to help the team to perform security engineering activities such as risk assessments. Every developer, not just a single person, should be aware of possible risks to the system and how they can be mitigated. A possible way of how this can be achieved is that several developers perform a risk assessment for each user story. We suggest that it is performed with the help of a security expert in order to achieve something like collective ownership. In this way many developers are aware of potential risks and understand which security controls have to be implemented in order to defend oneself against them. After a few iterations the developers should be able to perform threat modeling or risk assessments without the help of a security expert [11].

The second option would be to find agile methods that assist developers with minimal security knowledge to perform a risk assessment or threat modeling and to help them to break non-functional security requirements down into several estimable tasks. SAFECODE templates as stated in Chapter 2 can therefore be used. Figure 4.1 shows an example of such a template. The first column illustrates the number of the template, whereas the second column describes a high level security requirement. The A, D and T in brackets in column two have the following meaning: A (Architect), D (Developer), T (Test). The third and fourth column describe possible security controls and the last column highlights a link to the "Common Weakness Enumeration" database [24]. This database enumerates several common weaknesses, and how one can prevent them as well as the most common failures. Additionally, CWE entries list several code examples according to a specific high level security requirement. They should be used in the following way [11]:

- Developers use the high level security-related story description (non-functional security requirement) in the second column in order to identify whether a specific template can be applied to a

functional user story or epic.

- Column three lists several tasks how such a requirement can be broken down into estimable tasks. Developers can use these tasks suggested in order to define their own for a specific user story.
- Furthermore, in order to fulfil these requirements they can perform the *fundamental practices* suggested in column four, which describe possible security controls and verification methods.
- For further investigations the link to the "Common Weakness Enumeration" database is provided. This database contains a detailed description to many common security weaknesses:
 - How to mitigate against them with code examples,
 - Which testing tools should be used, for example: fuzzy testing or static- and dynamic code analysis, and
 - How to plan to mitigate against them with a formal explanation of what needs to be done in which software development phase.

No.	Security-focused story	Backlog task(s)	SAFECode Fundamental Practice(s)	CWE-ID
1	As a(n) architect/ developer, I want to ensure AND as QA, I want to verify allocation of resources within limits or throttling	<p>[A] Clearly identify resources. A few examples:</p> <ul style="list-style-type: none"> • Number of simultaneous connections to an application on a web server from same user or from different users • File size that can be uploaded • Maximum number of files that can be uploaded to a file system folder <p>[A/D] Define limits on resource allocation.</p> <p>[T] Conduct performance/stress testing to ensure that the numbers chosen are realistic (i.e. backed by data).</p> <p>[A/D/T] Define and test system behavior for correctness when limits are exceeded. A few examples:</p> <ul style="list-style-type: none"> • Rejecting new connection requests • Preventing simultaneous connection requests from the same user/IP, etc. • Preventing users from uploading files greater than a specific size, e.g., 2 MB • Archiving data in file upload folder when a specific limit is reached to prevent file system exhaustion 	<ul style="list-style-type: none"> • Validate Input and Output to Mitigate Common Vulnerabilities • Perform Fuzz/ Robustness Testing 	CWE-770

Figure 4.1: Example of a security-related user story of SAFECode. Adopted from [11].

Furthermore, [11] proposes that all user stories should have a security debt value and if they are postponed or skipped, the debt of a Scrum team increases. If this value exceeds a certain predefined norm, the team has to perform a security Sprint. In this security Sprint all these security-related user stories must be developed.

Proposal of how PA 09 ("Provide Security Input") and PA 10 ("Specify Security Needs") can be integrated into Scrum

From our point of view, both options should be integrated into Scrum. Developers should be trained in security and one risk modeller should be added to each Scrum team. The risk modeller's role is to help the development team to perform security engineering activities such as risk assessments. Furthermore, developers should use SAFECode's security-related user story templates to identify security requirements of user stories, because, in our opinion, these templates are powerful tools supporting the process to enumerate all possible security implications of a functional user story or epic. However, not every security-related user story should have a security debt value. We claim that tasks of high prioritized requirements should be mandatory tasks of a functional user story and should be added as acceptance criteria. Thus, high prioritized security concerns cannot easily be forgotten in the development.

4.1.2.2 PA 07: Coordination of Security

Proposal of how PA 07 ("Coordination of Security") can be integrated into Scrum

As stated in [88], a Scrum team is self-organizing and consists of a maximum of nine people. Team communication is achieved by Daily Scrum meetings, Sprint Planning meetings, Product Backlog Refinement meetings, Sprint Reviews and frequent face-to-face interactions. Hence, we suggest that the coordination of security issues can be accomplished when considering the following issues:

- A security expert and developers perform a risk assessment for each user story in backlog refinement meetings. SAFECode templates and the CWE links included can be used to assist the risk assessment.
- A product owner and a security expert use the results of the risk assessment and define security requirements for each user story or epic in cooperation with the customer.
- These security requirements are then formally written down in security-related user stories and attached to functional ones.
- A product owner or security expert report them informally to the developers and formally with the help of security related user stories.
- In another backlog refinement meeting the developers break down the user stories in the Scrum backlog into estimable tasks and think about possible security controls.
- Further, in Sprint planning meetings the whole team prioritizes all security requirements and discusses which security controls should be implemented.
- Next, developers pick the high prioritized security controls in daily standups and start to implement them.
- If problems occur when developers are implementing security controls, the issues can be discussed with the whole team owing to the small team size, self-organization and frequent face-to-face interactions. Furthermore, if very difficult problems occur another backlog refinement meeting can be held to re-prioritize the user story.

4.1.3 Assurance Process

Summary of the SSE-CMM's assurance process

The assurance process consists of two PAs (PA 06 and PA 11). PA 06 ("Build assurance argument") focuses on providing evidence that all defined security goals are actually accomplished. This process area should generate security requirements deriving from the goals which ultimately prove that the software developed is actually secure. PA 11 ("Verify and Validate Security") aims to verify if the implemented solution is precisely what was planned and to validate that all defined security goals are accomplished.

4.1.3.1 PA 06 ("Build assurance argument")

Discussion of how PA 06 ("Build assurance argument") can be integrated into Scrum

According to SSE-CMM, it is essential to generate a requirements traceability matrix where requirements are mapped to security goals which were defined by the customer [39]. The aim of these methods is to verify whether all specified security requirements fulfil the defined goals. Unfortunately, most of these methods directly contradict principles and practices of agile development [18].

Proposal of how PA 06 ("Build assurance argument") can be integrated into Scrum

The customer can be involved in defining all high and low-level requirements, corresponding to acceptance criteria and test procedures [39]. To be more specific, first, the team should identify security policies, standards and compliance regulations and should derive requirements from them with the help of the customer [78, p. 16–17]. Further, we recommend that requirements should be derived from identified risks of threat modeling and be discussed with the customer, because they can help to prioritize the threats. As a second step the requirements should be validated with several different testing techniques. In Scrum, testing should be done early and often and can therefore fit into agile [87]. For this reason methods have to be shown that fulfil these requirements. Section 4.1.3.2 discusses some of them elaborately. The involvement of the customer is agile-friendly, as stated in the agile manifesto. Moreover, this would result in fulfilling the goal partially to generate security requirements, deriving from the goals which prove the customer that the software being developed is actually secure. The security requirements would reflect the customer's expectations to the software. However, since many customers are almost likely no security specialists and only have limited knowledge about security, this can be a challenging task.

4.1.3.2 PA 11: Verify and Validate Security

Discussion of how PA 11 ("Verify and Validate Security") can be integrated into Scrum

In this section we discuss several security assurance methods and techniques that Beznosov and Kruchten [18] suggests possibly fitting into agile.

Internal reviews of design and implementation Beznosov and Kruchten [18] claim that the "agile friendliest" way is to use internal reviews of design and implementation. In this method a security expert either reviews the source code manually or with the help of static code analysis tools. However, this requires that one has to know what to look for in the code. Without security expertise and domain specific knowledge many failures can be overseen. Automatic tools, such as static analysis, help to first identify possible code errors, but to filter false positives one has to have security expertise too. Nevertheless, in the opinion of OWASP [78, p. 13–14] it is a fast and effective way for identifying vulnerabilities. They also state that in contrast to other methods, such as penetration testing static code analysis tools are capable of identifying, for example, access control problems, backdoors and cryptographic weaknesses. According to Schwaber [87], in Scrum, code reviews are important, that is why we also think that this methodology should be part of an agile development process. Software is developed in increments and therefore only small units of code have to be reviewed each Sprint. We suggest that a security expert or software architect reviews the code of each user story in the implementation phase. In this way, many security flaws can be identified before it is tested.

Testing

According to Schwaber [87], testing should be done early and often in Scrum and hence is an important technique of assuring that software is secure. Beznosov and Kruchten [18] propose that the following testing methods and techniques can help to assure that security requirements of agile projects are fulfilled in an agile-friendly way (see Section 3.7.1 for their definitions):

1. Security testing,
2. Vulnerability- and penetration testing, and
3. Security static analysis.

We discuss whether these propositions can be used in agile in the next paragraphs.

Penetration testing Penetration testing of web applications is a special type of security testing and is typically used in the verification phase of a project [78, p. 16]. It can be performed manually or automatically. On the one hand, manual penetration testing is very time consuming, because pentesters often write their own code and scripts to exploit vulnerabilities. They investigate where vulnerabilities can occur in the code manually or automatically with the use of tools [49], for example, with the help of a web vulnerability scanner, such as Acunetix. So called, "all-in-one" automatic tools, such as Metasploit [53] use standard attack patterns of common and publicly-known attacks in order to automatically attack a system and to consequently exploit vulnerabilities [49]. However, a pentester who performs manual penetration testing is able to execute many complicated up-to-date attacks which might not be included in the set of attack patterns of "all-in-one" automatic tools. Unfortunately, this obviously requires a security expert who knows several attack patterns. The quality of the penetration testing therefore in this case depends on the hacking skills of the pentester. On the other hand, automatic "all-in-one" utilities, are much more time efficient than the manual approach. Nevertheless, many hackers can use much more complicated attacks than already known basic ones. Hence, not all potential assaults on information systems can be tested with these tools, but it is still vital that the used kind of penetration testing is as time efficient as possible in order to fit into agile.

Whitebox testing Another kind of security testing is whitebox testing, such as unit and integration testing. The agile practice, test-driven development [8], focuses on this technique, thus we claim that it is agile. It is performed in the implementation phase and reveals, for example, side effects of the code and random typographical errors. However, missing functionality cannot be discovered, because it only tests existing code as it is. Although in theory it is desirable that all parts of software are tested with this kind of technique, we reckon that it is hard to achieve in practice to test all parts of the software, because it obviously needs a considerable amount of time. We believe that it is important to test with whitebox testing as much as possible in order to assure that the software has no sideeffects.

Static code analysis tools Even though static code analysis tools can be helpful to identify coding errors, they deliver many false positives and thus much security expertise and manual effort is needed to filter them [78, p. 16]. Moreover, Beznosov and Kruchten [18] suggests obvious high-level security methods that can be used in agile, such as the use of high level programming languages and tools, use of secure design principles and adherence to implementation standards. Nevertheless, the first point depends on the project and the company. In our case, web applications are mainly programmed in high-level programming languages, such as JAVA, .NET, and PHP. The last two points can be achieved by training developers with regard to security. We also think that these points are very important, because if security guidelines are mandatory, then, the developers' awareness of security is higher and security is implicitly taken into consideration when programming. Nevertheless, to determine in how far that can be done is not the aim of this thesis.

Proposal of how PA 11 ("Verify and Validate Security") can be integrated into Scrum

As can be seen above, several methods, techniques and tools exist that can be used to ensure security, however, many of them require security expertise in order to perform them in an effective way. In our opinion, it is important to hire security experts that perform them or to train employees so that they would

be capable of using them in a productive way. We also think that there is no single ideal security testing approach and that security assurance should be achieved by several methods instead of performing only one technique. Developers should write as many unit, integration and system tests as possible. Code reviews should be performed with the help of static code analysis tools for each user story. Next, a vulnerability scanner should be used each Sprint to scan for vulnerabilities in the code and a pentester should perform manual penetration tests regularly, but not every Sprint. However, we do not know how agile friendly our approach is, hence we perform in Chapter 5 an analysis of several techniques and methods in order to show how agile each is.

4.1.4 Summary of this Chapter

Table 4.1 summarizes the results of our performed analysis in Chapter 4. This table lists every Process Area of the SSE-CMM and outlines whether Scrum handles a specific PA of the SSE-CMM. The "Process Area" column describes the process area of SSE-CMM, whereas the "Does standard Scrum support a specific PA?" column is dedicated to the purpose to check whether Scrum handles a particular PA and describes how Scrum can handle a concrete PA.

In summary, Scrum does only support one PA, partially supports three PAs, and does not support five PAs. It is also important to mention here that we do not consider PA 01 ("Administer Security Controls") and PA 08 ("Monitor Security Posture"). The focus on PA01 and PA08 lies on project management, staffing and organisational practices, which are not specific security engineering tasks. Thus, we did not consider these process areas.

In Chapter 5 we evaluate how agile security activities, practices, tools, and methods of Microsoft SDL, Cigital Touchpoint, SREP, and NIST 800-64 are. We evaluate them in order to know which specific methods, techniques, and tools can be used to support all PAs of the SSE-CMM which are not or only partially fulfilled by Scrum.

Process Area (PA)	Does standard Scrum support a specific PA?
PA 01: Administer Security Controls	Not considered.
PA 02: Assess Impact	Scrum does not support this PA.
PA 03: Assess Security Risk	Scrum does not support this PA.
PA 04: Assess Threat	Scrum does not support this PA.
PA 05: Assess Vulnerability	Scrum does not support this PA.
PA 06: Build Assurance Argument	Scrum does not support this PA. Scrum does not specify that for each software increment security requirements have to be defined. To ensure that the software meets the expectations of the customer in regard of security, the customer and a security expert can be involved in defining security requirements and acceptance criteria of a software increment. With assurance methods of PA11 it has to be assured that all security requirements defined are met.
PA 07: Coordinate Security	Scrum supports this PA. Security can be coordinated via small team communication, self-organized teams, Daily Standup meetings, Product Backlog Refinement meetings, Sprint Planning meetings, and Sprint Review meetings.
PA 08: Monitor Security Posture	Not considered.

PA 09: Provide Security Input	<p>Scrum partially supports this PA.</p> <p>If developers have security expertise, security considerations can be included in user story refinements and in code reviews. Developers can use SAFECODE's security-related user stories to define security requirements.</p>
PA 10: Specify Security Needs	<p>Scrum partially supports this PA.</p> <p>To define security requirements of a user story so-called security-related user stories can be defined by Product Owner, customer and developers and defined as acceptance criteria for the corresponding functional user story. SAFECODE's security related user story templates should be used to split security-related user stories into estimable tasks.</p> <p>Furthermore, a security expert should be added in the refinement of user stories and Sprint Planning meetings to help the developers to perform security engineering activities, such as risk assessments.</p>
PA 11: Verify and Validate Security	<p>Scrum partially supports this PA.</p> <p>According to Scrum, testing and code reviews should be performed. In code reviews security flaws in the source code can be identified by manually reviewing it. By writing security tests it can be verified whether the source code meets the defined security requirements. Security testing methods which can, from our point of view, be used are: Web vulnerability scanning, and whitebox testing, such as unit- and system testing</p>

Table 4.1: How Scrum handles SSE-CMM's PAs

Chapter 5

Analysis of Existing Well-Established Security Activities and Tools

In this Chapter we analyse activities of the following SE processes with regard to how “agile-friendly” they are:

- NIST 800-64 [47],
- Security Requirements Engineering Process (SREP) [52] which is based on Common Criteria,
- Digital Touchpoint [50], and
- Microsoft SDL [34].

In Chapter 4 we analysed which high-level security practices are necessary to create a secure Scrum process and we evaluated whether Scrum supports some of these security practices. We performed this analysis in order to know which practices are required to create a secure Scrum process.

The aim of this Chapter is to identify specific activities, methods, tools, and techniques of well-established SE processes that we can integrate in our secure Scrum process and which cover SSE-CMM’s practices. We call these activities, methods, tools, and techniques throughout this document *security activities* to simplify the writing. We only analyse *security activities* with the focus on specifying security requirements, designing and implementing of secure software, and verifying whether it is actually secure. Therefore, we do not consider activities which aim to train employee in security or to deploy or release the software.

First, we outline which method we use, after which we briefly explain the activities which we analyse and then we present the results. The analysis is in Annex A. Finally, we compare our results with the results of a related work. To analyse the agility degree of an activity, methodology, technique, tool, or practice we use the method described in [46]. According to the authors of [46], an activity is "agile friendly", if all *agile features* listed below are fulfilled:

- simplicity,
- high customer interaction,
- free of modeling and documentation,
- tolerant to requirement changes (not much work has to be redone on requirement changes),
- minimal speed of execution,
- people oriented,
- informality,

- iterative, and
- high flexibility.

The authors of [46] extracted these *agile features* from the agile manifesto, but they never stated what their exact meaning is.

We claim that the meaning of the agile features, *simplicity*, *change tolerance*, *people orientation*, *the ability that it can be performed iterative*, *high flexibility*, and *informality* is not obvious and has therefore to be defined. In contrast, the meaning of *high customer interaction*, *minimal speed of execution*, and *free of modelling and documentation* is, in our opinion, clear. Hence, we defined their meanings ourselves, which we describe in greater detail below. Furthermore, we decided to remove the *high customer interaction* feature, because we interpret that the agile value that customer collaboration should be over contract negotiations [5] that the customer should only be involved in planning tasks. We reckon that mandatory customer interaction would be a bottleneck in all other activities, such as static- and dynamic code analysis, code reviews, develop security documentation and penetration testing, due to the fact that customers do not always have time for quick responses.

Keramati and Mirian-Hosseiniabadi [46] propose that in order to measure how agile an activity is, for every feature a grade in the interval [0, 5] is assigned. The agility degree of an activity is the sum of all measured feature grades. If this value is high, the activity can be integrated into Scrum and - if not that means integrating it would be difficult, because it is not “agile-friendly”. However, the authors of [46] did not specify the threshold of the agility degree to be “agile-friendly”. The specific meaning of each feature is needed in order to measure the agility degree.

5.1 Our approach

5.1.1 Our interpretation of agile features

We think that this method is good to start with, but these features have to be defined in greater detail. Our interpretation of them is as follows:

- **simplicity:** What kind of security expertise must the person who performs an activity have in order to complete it in an effective way?
- **free of modeling and documentation:** Does the work not only consist of documentation and modelling? Is something implemented or tested?
- **tolerant to requirement changes:** Is it necessary to wait for the completion of other activities in order to be able to complete an activity? How much has to be redone on small or medium changes requirement changes? (Small or medium requirement changes means that requirements of a software increment that should be implemented do not change completely.)
- **minimal speed of execution:** How long does it take to perform this activity? In case of a tool: Can it be performed over-night?
- **people oriented:** Are people involved who have to think about problems in brain storming sessions? Does an activity require much people interactions? Or is it quite method-based and has to be strictly performed step by step according to a method.
- **informality:** Is this activity needed to better understand security requirements of a software increment? Is it needed to ensure that a software increment is secure?
- **iterative:** Does this activity consists of several steps? Can the work be divided among several people? Should it be performed iteratively?

- **high flexibility:** Is it possible to skip some steps, without being ineffective, due to, for example, time reasons? Can steps be replaced by other ones which are more time-efficient as the original ones? Should all steps be performed strictly as described to be effective?

5.1.2 Our metric

We created a metric in order to define how these grades can be measured. Figure 5.1 and Figure 5.2 illustrate our metric. We defined the grades and use them to roughly estimate the agility degree of activities. It is essential to mention that the metric is based on the context of a user story or epic which is implemented in about two weeks. Unfortunately, it is not possible to measure them exactly, nevertheless we discuss each activity and why we chose a specific grade for the evaluation. We claim that it is not important that an activity can be divided if it can be quickly performed. We think that the ability that it can be performed iterative is only necessary if the execution time is high. Hence, we created a second metric which describes the correlation of the "iterative / dividable" and "speed of execution" feature. Figure 5.3 displays this correlation.

The maximum points an activity can obtain is 40. If one has more than 21 points (>55% of the maximal obtainable points) we say that it is ("agile compatible"). We chose 21 points, because then an activity obtained definitely more than half of the maximal achievable points. If an activity has more than 26 points (>67,5% of the maximal obtainable points) we claim that it is "agile friendly".

5.1.3 Validation of results

In order to make the results more reliable we reinforce them with an existing case study of [13]. This case study is based on a survey in which several employees of the company Ericsson AB, which develops telecom products, were interviewed how cost-effective and beneficial an activity of Cigital Touchpoint, Common Criteria and Microsoft SDL is in an agile project in regard to security. Each role in Ericsson AB's development process was represented by two or more employees. For each activity they assigned either a "-" (-1 for our calculation), " " (0 for our calculation), or "+" (1 for our calculation) on the cost or benefit. A "-" means that the costs are high and that the benefit is low, whereas a "+" states that the costs are low and that the benefit is high. The " ", describes that cost or benefit are tolerable. We take their results of Cigital Touchpoint and Microsoft SDL to validate our results. To reinforce our results of SREP we take their results of Common Criteria, due to SREP is based on Common Criteria and therefore the activities are very similar. Unfortunately, they did not evaluate NIST 800-64, thus, we cannot validate these results. If the sum of the benefit and cost-effectiveness of an activity is at minimum zero our grading stays the same. However, if for any reason that should not be the case, the activity is considered to be not "agile friendly". In this case we discuss in this Chapter why they evaluated it as not "agile friendly" and why we think that it can actually be "agile friendly".

Criterion	0	1	2
Simplicity	Security Expert Very complex (Security expert which can hack software systems and has many practical knowledge)	Senior Security Specialist Complex (Thinking about vulnerabilities, threats, attacks without tool support, knows many attacks). Can design security architectures.	Junior Software Specialist Medium Complexity (Identification of vulnerabilities, threats, attacks with tool support, knows common attacks)
Free of Modeling and Documentation	Task is only documentation effort and no Implementation or testing.	More documentation effort than Implementation or testing.	Documentation and implementation or testing effort is equal.
Change Tolerate *	If changes appear: A very high amount (90-100%) additional effort than in first iteration. Or The activity depends on results of more than two activities.	If changes appear: A high amount of the activity has to performed again. Or The activity depends on results of one (speed of execution is lower than 4) or two activities.	If changes appear: Definitely more than 50% of the activity has to performed again. The activity does depend on results of no activity or one (speed of execution higher than 3) .
Speed of Execution **	a sprint	16 - 40 hours	8 - 16 hours
Informality	No informality: No Information for securing the software is gained.	Very low informality: Only few knowledge about requirements is gained or improves assurance of security only a bit better	Low Informality: Provides knowledge to better understand requirements or improve assurance of secure software.
People Orientation	Formal and method based.	Small amount of people orientation. Important part is only formal and method based.	More formal and method based than people oriented.
Iterative / Dividable	Just performed once.	A small amount of subtasks can be performed iterative Main part just performed once.	About 50% of all subtasks can be performed iterative.
Flexibility	It must be performed as described or all planned tasks have to be made in order to be effective.	The most parts of the activity have to be performed or most planned tasks have to performed.	Only 50% parts of the activity have to be performed or only 50% of all planned tasks have to be performed.
* Small or medium changes means that software requirements of a user story do not change completely. Software requirements are dropped or several are added. At most 50% of all software requirements change.			
** Metric is based in the context of a user story or epic which is implemented in about 2 weeks.			

Figure 5.1: Metric for evaluating the support of agile values (grades 0-2).

Criterion	3	4	5
Simplicity	Developer with Security Interest Low Complexity (Knows basic principles of security and the OWASP Top 10)	Normal Developer Very low complexity (Does not know the OWASP Top 10, but has domain specific knowledge)	Person without technical background Very easy (Person with not many technical expertise)
Free of Modeling and Documentation	More implementation or testing effort than documentation	Small amount of documentation effort	No documentation needed
Change Tolerate *	If changes appear: about 50% of the activity has to be performed again. The activity does not depend on results of other activities.	If changes appear: Less than 50% of the activity has to be performed. The activity does not depend on results of other activities.	If changes appear: Only several parts of the activity have to be performed again. The activity does not depend on results of other activities.
Speed of Execution **	4 - 8 hours	2 - 4 hours	1 - 2 hours
Informality	Medium Informality: Needed to better understand requirements or improves assurance of security.	High Informality: Highly recommended to better understand requirements or immensely improves assurance of security.	Very High informality: Without this activity requirements can not be understood or security can not be assured or not be planned.
People Orientation	Formal and method based is equal.	More people oriented than formal and method based.	Only people oriented.
Iterative / Dividable	Definitely more than 50% of all subtasks can be performed iterative	A high amount of subtasks can be performed iterative. Or Iterative structure can be used, but each step is quite heavy-weight.	Iterative Structure can be used and each step is quite light-weight.
Flexibility	Less than 50% parts of the activity have to be performed or less than 50% of the planned tasks have to be performed.	Only several parts of the activity must be performed or only several of the tasks have to be performed.	Cherry picking of subtasks of this activity is possible or cherry picking of the planned tasks can be performed.
* Small or medium changes means that software requirements of a user story do not change completely. Software requirements are dropped or several are added. At most 50% of all software requirements change.			
** Metric is based in the context of a user story or epic which is implemented in 2 weeks			

Figure 5.2: Metric for evaluating the support of agile values (grades 3-5).

(Speed of Execution) / (Iterative / Dividable)	0	1	2	3	4	5
0	0	0	0	0	3	5
1	1	1	1	1	3	5
2	2	2	2	2	4	5
3	3	3	3	3	4	5
4	4	4	4	4	4	5
5	5	5	5	5	5	5

Figure 5.3: Metric for defining how the speed of execution depends on iterative-/dividability.

5.2 Activities and tools assessment

In this Section we list all *security activities* which we extracted from Microsoft SDL, Cigital Touchpoint, SREP, and NIST 800-64 and explain each. In this Chapter we only list and explain each security activity and show the results of this analysis. In Annex A we describe for each security activity why we gave it a specific grade in a particular agile feature.

5.2.1 Microsoft SDL

We extracted the following activities from [34] from the phases requirements, design, implementation, and verification.

5.2.1.1 Security Requirement Analysis

Howard and Lipner [34] do not specify in detail how a security requirement analysis should be performed. They stated that use scenarios should be defined and that security experts should conduct a security and privacy analysis for defining minimal security and privacy requirements and provide therefore a security requirements questionnaire. Additionally, several security requirements should be derived from the results of secure code reviews. Thus, this task requires brainstorming about where security- and privacy requirements and use cases are defined.

5.2.1.2 Role Matrix

The goal of this activity is to identify all user roles of the application and to determine their access restrictions to functionalities and assets of the application.

5.2.1.3 Design Requirements

The goal of this activity is to model a secure design graphically, which represents the architecture of the system. UML, class, sequence diagrams, or several other modeling techniques can be used for this activity.

5.2.1.4 Cost analysis (By conducting a Product Risk Assessment)

According to [34, p. 93–99], the goal of this activity is to roughly estimate the cost, time and effort of a software project. For this reason, one has to analyse which parts of the project require threat modeling, security design reviews, and penetration testing. Additionally, the scope of fuzz testing is defined and whether private information is processed, stored or transmitted by the software. This is achieved by doing a security risk assessment and a privacy impact rating.

Security Risk Assessment In this step a security team fills out a questionnaire which focuses on questions regarding possible vulnerabilities of a system and which security components are needed. For example, a question can be: “Is authentication or authorization needed?”. As next step, the answers are analysed and based on general defined rules and on the expertise of the analysts decisions are made, for which components of the software, for example, threat modeling is needed.

Privacy Impact rating In this step a security or privacy team analyses how sensitive the data are that the software processes, transmits, or stores. They should be categorized into three groups: anonymous data, personally identifiable information and sensitive information. This information also helps to roughly identify how much effort is needed to secure the information.

5.2.1.5 Threat Modeling

See Section 3.4.5 for a detailed description.

5.2.1.6 Attack Surface Analysis / Reduction

The goal of this activity is to reduce the attack surface of the software [34, p. 78–89]. In order to do so the principle of least privilege should be applied to all processes and services of the application. Code or services that are not needed should be shut down or removed and authentication and access restrictions should be implemented. Further, data flow diagrams and graphic presentations of the code should be identified that does not require authentication. Finally, access to network endpoints should be limited, and entry points which are not needed should be removed.

5.2.1.7 Static Analysis

We already described this kind of tool in Section 3.7.

5.2.1.8 Dynamic Analysis

See Section 3.7 for a detailed description.

5.2.1.9 Fuzz Testing

We already explained this kind of testing in Section 3.7.7.

5.2.1.10 Code Review

The aim of this activity is to identify security flaws in the source code by manually reviewing it.

5.2.2 Digital Touchpoint

We deduced the activities below from [50]:

5.2.2.1 Security Requirement Analysis

The authors of [50] list three possibilities of how security requirements can be identified. In the first they should be defined through abuse cases, which are described in the activity below. The second aims to derive them from three different sources: laws and regulations, commercial considerations and contractual obligations. They should then be prioritized by severity. In the third they are derived from risk assessment results. In this activity we only consider the second option and we do not take abuse cases or risk assessments into account since we treat them as separate activities.

5.2.2.2 Abuse Cases

For defining abuse cases the authors of [50] state that four steps are necessary. First, groups of attackers (threat agents) have to be identified. As second step, so called anti-requirements have to be determined, which state unacceptable outcomes of the software. The third step consists of identifying specific attack models that illustrate how an unacceptable outcome can be achieved. Finally, mitigation strategies are established in order to defend the application against these attack models. An abuse case is a combination of these steps and describes how one can mitigate that a specific threat agent successfully performs a particular attack model in order to exploit a vulnerability to cause an unexpected behavior of the software.

5.2.2.3 Architectural Risk Analysis

An architectural risk analysis is a part of risk analysis and focuses on the identification of design flaws. It includes the following steps [50, p. 139–170]:

First a one page graphical overview of the system should be drawn and then the following three high level steps performed:

1. Attack resistance analysis First, security flaws are identified with checklists and historical risks from previous analyses. Second, attack patterns are mapped to abuse cases. Third, risks are determined with the help of checklists. At last, attacks are demonstrated with, for example, exploit graphs.

2. Ambiguity analysis (needs at minimum two analysts) Each analyst performs his own risk analysis and then the findings are discussed with everyone and are “unified”.

3. Weakness analysis Weaknesses in the dependencies of the software, such as frameworks or browsers or servers are analysed.

5.2.2.4 Manual or semi-automatic Penetration Testing, Automatic Penetration Testing, Static Analysis, and Red Team Testing

We already described these testing techniques in Section 3.7.

5.2.2.5 Risk Based Testing

The authors of [50] suggest using manual testing based on risks. In risk-based testing [7] test cases are prioritized by risk, which are determined prior in a risk assessment. In this way the high-risk components are tested first and the low-risk ones at the end of the testing process.

5.2.3 SREP

From [52] we extrapolated the following activities:

5.2.3.1 Security Requirement Analysis

SREP is a Security Requirements Engineering Process. The aim of such a process is to integrate the defining of security requirements into a software engineering process. An essential part of it is to elicit and specify security requirements, which is to goal of this activity. We combined three activities of SREP into a single activity (Security Requirement Analysis), because each of them focuses on the identification and specification of security requirements. These activities are (we extracted them from [52]):

Identify security objectives and dependencies In this activity, requirements are derived from the organizations security policies, legal requirements, critical assets, (can be taken from the results of the “Critical Assets” activity of SREP which we describe below), and likely attacker types.

Elicit security requirements Here, requirements are derived from the identified threats of the “Risk analysis” activity of SREP which we describe below.

Categorize and prioritize requirements This activity focuses on categorizing and prioritizing the already identified security requirements.

5.2.3.2 Agree on Definitions

In the Agree on Definition activity the customer defines with the organization, which security definitions and security policies should be established. The policies are mainly taken from ISO/IEC- and IEEE standards. Additionally, the security vision of the project is documented in a so called Vision Document. The authors of [52] state that this activity is performed in the inception phase of a project and is not revisited in later phases.

5.2.3.3 Risk Analyses

The authors of [52] stated that a risk analysis should be performed in their process, however, they did not specify a particular methodology, but rather listed some existing ones, which can be used. Among others, they suggested CRAMM [102]. We chose it, because it is a managerial level risk assessment, which is based on meetings, interviews, and questionnaires. This type of risk assessment does not take specific vulnerabilities which can be detected through; for example, threat modeling, static code analysis, or penetration testing into account. Thus, we thought that it can be quite light-weight and can be an alternative to heavy-weight methodologies, such as Microsoft's Threat Modeling. For this methodology a tool is provided. This tool contains several structured question in regard of security. Reviewers should ask employee these structured question and have then to enter the answers into the program. The employees who are interviewed can be, for example, management personnel, asset owners, or employees with security expertise in different categories. Based on the answers the program then calculates the risk of threats and countermeasures. The methodology consists of the following steps:

1. Initiation A meeting with the organization's management and the CRAMM reviewer are held. The goal of this meeting is to define the scope of the assessment and to identify the interviewees.

2. Identification and Valuation of Assets Assets are identified. This can be achieved, for example, through the "Critical Assets" activity, which is described below. As a next step, interviews with data owners are held in order to measure the importance of each asset.

3. Threat and Vulnerability Assessment Assets are grouped and then only the most important ones are picked for identifying threats, in order to be time-effective. To support the identification CRAMM provides threat/asset group and threat/impact tables. Thus, only questions to threats which can harm the most important assets should be questioned. Threats and vulnerabilities are then assessed by interviewing support personnel with the help of structured questionnaires. The answers are entered in the program.

4. Risk Calculation The tool calculates the risk for each threat and suggests security controls to prevent these risks.

5.2.3.4 Critical Assets

The goal of the Critical Asset activity is to identify critical or vulnerable assets. The assets can be identified through the help of lists of assets, which are grouped by domain or through similar profiles. Moreover, they can be derived from functional requirements or through interviews with stakeholders.

5.2.3.5 Identify Threats and Develop Artifacts

The aim of this activity is to identify threats, which have not already been determined and also ones which does not target already identified assets. In order to achieve this, artifacts, such as, UMLSec

diagrams, attack trees, or abuse cases should be developed for deriving new threats from them. These identified threats, created artifacts, and assumptions should then be documented in a so called Security Problem Definition Document.

5.2.3.6 Requirement Inspection

The aim of a Requirement Inspection is to validate the security requirements. For this reason all the artifacts, which were produced in all the described activities above are reviewed. Furthermore, SSE-CMM, Common Criteria assurance requirements and the EALs (Evaluation Assurance Level of Common Criteria) are used to verify whether the software is secure (we already described SSE-CMM and Common Criteria in Section 3.6 and Section 3.5).

5.2.3.7 Repository Improvement

Repository Improvement focuses on identifying threats or requirements (model elements) which are likely to occur in future development and to document them.

5.2.4 NIST 800-64

We took activities from the initiation and development/acquisition phases of [47]. We did not consider the other three phases, because they focus on activities when the software is already released. The authors of [47] linked for further details of several activities to other NIST Special Publications. Therefore, in several activities we also cite from other Special Publications.

5.2.4.1 Initiate Project Security Planning

According to [47, p.14–15] this activity consists of the following steps: (1) Identification of key Security Roles, (2) Stakeholder Security Integration Awareness, (3) Initial Project Planning. First the key security roles are identified. Second, security requirements, assumptions, testing and assessment techniques, and also possible implications, and more are explained to all stake holders. The aim is that all stake holders have a basic understanding of security-related topics of the project. At last, initial security milestones are identified.

5.2.4.2 Categorize Information System

The goal of this activity is to identify information systems and information (assets), such as financial data, to categorize them into information types, and to evaluate their impact on cyber-attacks and consists of the following steps [92, p. 13]:

1. **Identify Information Types.** With the help of provided lists of several information type categories, such as, for example, Health Care Administration, the information type of all information systems and assets are identified.
2. **Select Provisional Impact Levels.** Next, the impact level (low, moderate, high) on integrity, availability and confidentiality on information system and assets are evaluated. The authors of [92] support this evaluation with a predefined metric that explains each impact level on a high-level.
3. **Review Provisional Impact Levels.**
4. **Adjust/Finalize Information Impact Levels.**
5. **Assign System Security Category.**

6. Finally, if the impact levels of all information system and assets are calculated the system security category is evaluated. It is an aggregate of the impact levels of all information system and assets.

5.2.4.3 Assess Business Impact

This activity is dividable into the following steps [93, p. 15–26]:

Determine Business Processes and Recovery Criticality Identify business processes and evaluate their impact on availability, integrity and confidentiality. Information systems can have several business processes. This analysis is executed in order to analyse the impact on information systems in greater detail. Furthermore, it is calculated how much it would the company cost if an information system is unavailable for a specific time-span.

Identify Resource Requirements Identify resources, which are needed to recover critical systems.

Identify System Resource Recovery Priorities First, preventive controls, such as frequent scheduled backups are identified. Second, availability impact level and corresponding backup strategies are defined. At last, costs, roles and responsibilities are identified.

5.2.4.4 Assess Privacy Impact

The expected outcome of this activity is a privacy impact assessment, where for every information system where sensitive information is processed, security controls are documented that mitigate the information from being exposed [47, p. 18]. In order to achieve this, the analysts should, for example, ask questions to address privacy considerations, such as “What type of information is collected on this information system?”, “Why is it stored?”, “Who has access to this information?”, “Which security controls can be used to ensure that this information is not exposed?” and document the answers [62]. Additionally, for identifying safeguards the activities “Asses risk to system” and “Select and Document Security Controls” of NIST 800-64 should therefore be used. We describe these activities below.

5.2.4.5 Assess Risk to System

The aim of this activity is to perform a risk assessment. We already described this activity in Section 3.4.7.1.

5.2.4.6 Select and Document Security Controls

As the name suggests, the goal of this activity is to select and document security controls, which consists according to [47, p. 23–24], of the following steps: (1) Common control identification, (2) Security baseline selection, (3) Tailoring of security baselines, (4) Supplementation of further security controls to mitigate threats which were identified in risk assessments. We describe the first three steps below. The authors of [47] linked for greater details of these steps to other documents and therefore we cite from these linked documents to explain them.

Common control identification This step is described in NIST SP 800-37 [63]. For each information system common controls (are security controls which are inherited from one or more organizational information systems) of the organization are identified and then documented [63, p. 24].

Security control selection (Security baseline selection and tailoring of security baselines)

These two steps are specified in the NIST SP 800-53 document [65]. According to NIST [65, p. 28–45], first, a security control baseline (high-level security control, such as authentication) is selected from the security control baseline list of the NIST SP 800-53 document based on the impact level of the information system. Second, based on risk assessment results the baseline security controls are tailored. The scope of them is defined and then more specific controls are selected from a security control catalog of the NIST SP 800-53 document. The aim of this step is to specify the selected baseline security controls on a low-level and to adjust them to fit with the conditions within the organization. They should meet the security requirements and mitigate against the threats identified from the risk assessment. As a next step, so called “overlays” can be created if needed. These overlays are tailored security baseline controls for community-wide use, such as public key infrastructure (PKI) or cloud services. Additionally, these overlays can be specialized security controls that address special cases, which are not mentioned in the security controls catalog of the NIST SP 800-53 document. Finally, the control selection process is documented in a so called System Security Plan. A security plan contains the requirements of a system and describes the security controls for meeting those requirements.

5.2.4.7 Design security architecture

The aim of this activity is to develop a secure design of the software and consists according to [47, p. 24–25] of the following steps:

1. Analyzing system security plans and security requirements,
2. Think about a secure design. Several steps which can be performed are:
 - (a) Built selected security controls of previous activities in,
 - (b) Cluster functionality together, distribute them, or build a layered structure,
 - (c) Consider external services and dependencies,
 - (d) Consider results of security tests, or audits for investigating potential security breaches.
3. Visualize it graphically and document it.

5.2.4.8 Conduct Testing

This activity is described in NIST SP 800-53A [66]. The aim of it is to plan and assess the implemented security- and privacy controls, which were selected in the "Select and Document Security Controls" activity described above. According to [66], the testing consists of the following steps:

1. Preparing for security and privacy control assessments In this step preparation steps, such as defining the scope of the security- and privacy control assessment, specifying time frames for the assessments, and collecting artifacts of previous activities in order to get an overview of the information being assessed and the security and privacy controls which are assessed are conducted.

2. Developing security and privacy assessment plans Based on the selected security- and privacy controls of previous activities appropriate assessment methods are selected from the assessment procedure catalog of the NIST SP 800-53A document. As a next step, they are tailored for the characteristics of the information system of the organization. We describe their tailoring process in greater detail below. For each security control selected from security control catalog of NIST SP 800-53 an appropriate assessment procedure is listed in the NIST SP 800-53A document. Further, the selected assessment methods are optimized. The optimization can be achieved, for example, through combining testing procedures which tests the same security controls family.

Tailoring process

In order to conduct testing the following methods exist: examination, interview and test. The examination process consists of reviewing specifications, such as system requirements, or mechanisms, such as software functionality, or activities, for example, system operations. The interview process consists of interviewing one or more person within the organizations with specialized knowledge in an area. The testing process consists of testing if a software, hardware, firmware or activity has the expected behaviour under specified conditions.

All three methods have the attributes depth and coverage. For each of them grades should be assigned (basic, focused, or comprehensive). Basic coverage testing uses only a representative sample of assessments whereas by the comprehensive testing a sufficiently large sample is tested. An example for basic testing is black box testing with, for example, a vulnerability scanner. Comprehensive testing is white box testing. NIST provides in [61] a list of best practice assessment cases. These assessment cases describe tailored (grades for the assessment procedure attributes are assigned) assessment procedures.

3. Conducting security and privacy control assessments In this step the security- and privacy controls are actually assessed as defined in the security assessment plan developed in the previous step. For example, penetration or unit testing is performed.

4. Analyzing assessment report results Here the results are analysed whether all requirements are met and, all risks are reduced or prevented as planned in the risk assessment and security control selection process. If for any reason that should not be the case, it will be decided whether further security controls have to be implemented, security controls have to be modified, or if the remaining risk is ignored.

5. Assessing security and privacy capabilities The authors of NIST SP 800-53A state that often one safeguard alone does not mitigate a threat and therefore a collection of them is needed (a so called capability). In this step these capabilities are assessed. A security or privacy capability is a set of mutually reinforcing security controls or respectively a collection of privacy controls. They are selected and grouped in such a way that they mitigate or reduce risk from a specific threat family, for example, to achieve secure remote authentication. Normally security controls are assessed one by one. However, if one or more security control of such a collection does fail to mitigate a threat that does not mandatory mean that the whole capability does fail. The assessors should know how these controls are working together and should assess them based on this knowledge. If a whole capability fails they should be able to analyse which specific controls fail in a so called root cause analysis.

5.3 Results of the evaluation

We analysed how agile all *security activities* described above are with the help of Metrics 5.1 and 5.2 and the results of Baca and Carlsson [13]. The results of Microsoft SDL can be seen in Figure 5.5, of Cigital Touchpoint in Figure 5.4, of NIST 800-64 in Figure 5.6 and of SREP in Figure 5.7. These Figures are displayed in table format. The rows represent *security activities*, the first nine columns agile features, the "Agility Degree" column the agility degree of a specific security activity, the "Evaluation" column if it can be integrated into Scrum. A 1 or -1 in the last two columns represent the benefit and cost according to the case study. A "agile compatible" in the last column means that this activity reached >55% of the maximal obtainable points, whereas a "agile friendly" means that an activity obtained >67,5% of the maximal obtainable points. We highlighted some rows in dark grey, and did not assign any agile feature grade. This means that we were not able to evaluate one or more of the agile features of this activity. For example, for pair programming we were not able to evaluate the speed of execution, change tolerance, and informality. However, there is one exception. In case of NIST 800-64's activity "Develop security

documentation" we highlighted the activity in dark grey and gave it zero points in all agile features, due to its aim is only to produce documentation, which directly contradicts the agile feature, that the activity should be free from documentation.

Agility Feature/ Security Activity	Simplicity	Free of Modeling and Documentation	Change Tolerate	Speed of Execution	Informality	People Orientation	Iterative / Divideable	Flexibility	Agility Degree	Evaluation	Cost Efficiency	Benefit
<i>According Case Study</i>												
Security Requirements Analysis	2	0	3	4	5	5	5	4	28	agile friendly	1	1
Abuse Cases	2	0	2	3	4	2	5	4	22	agile compatible	0	1
Risk Analyses	1	0	2	2	4	3	4	4	20	agile unfriendly	-1	0
Penetration Testing	1	4	2	2	4	3	4	4	24	agile compatible	-1	1
Static Analysis	3	3	5	5	3	2	5	1	27	agile friendly	1	0
Red Team Testing	0	4	3	1	5	4	4	5	26	agile compatible	-1	0
Risk Based Testing	2	2	2	2	4	4	4	4	24	agile compatible	-1	0

Figure 5.4: Results of security activity analysis of Cigital Touchpoint in regard to how agile they are. [Image was taken by the author of this document]

Agility Feature/ Security Activity	Simplicity	Free of Modeling and Documentation	Change Tolerate	Speed of Execution	Informality	People Orientation	Iterative / Divideable	Flexibility	Agility Degree	Evaluation	Cost Efficiency	Benefit
<i>According Case Study</i>												
Security Requirements Analysis	2	0	3	4	5	4	5	4	27	agile friendly	1	1
Role Matrix	5	0	3	4	4	4	5	4	29	agile friendly	1	1
Design Requirements	1	0	2	2	4	3	4	4	20	agile unfriendly	-1	-1
Quality Gates									-	agile friendly	-1	0
Cost Analysis	2	0	0	4	3	4	4	2	19	agile unfriendly	-1	-1
Threat Modeling	1	0	2	2	4	2	4	3	18	agile unfriendly	-1	-1
Attack Surface Reduction	2	3	3	3	3	2	4	3	23	agile compatible	-1	-1
Security Tools									0		-1	0
Static Analysis	3	3	5	5	3	2	5	1	27	agile friendly	1	0
Dynamic Analysis	4	4	5	5	2	2	5	1	28	agile friendly	0	0
Fuzzy Testing	3	3	5	3	3	2	5	4	28	agile friendly	-1	0
Code Review	2	4	3	5	4	4	5	5	32	agile friendly	-1	0

Figure 5.5: Results of security activity analysis of Microsoft SDL in regard to how agile they are. [Image was taken by the author of this document]

Agility Feature/ Security Activity	Simplicity	Free of Modeling and Documentation	Change Tolerate	Speed of Execution	Informality	People Orientation	Iterative / Divideable	Flexibility	Agility Degree	Evaluation
Initiate Project Security Planning	1	0	4	5	5	5	5	3	28	agile friendly
Categorize Information System	2	0	4	4	4	3	4	1	22	agile compatible
Assess Business Impact	5	0	3	3	4	1	4	1	21	agile unfriendly
Assess Privacy Impact	3	0	4	4	4	4	5	1	25	agile compatible
Assess Risk to System	2	0	2	2	4	2	4	3	19	agile unfriendly
Select and Document Security Controls	2	0	1	3	4	2	5	3	20	agile unfriendly
Design security architecture	1	0	1	5	4	3	4	2	20	agile unfriendly
Engineer in Security and Develop Controls									-	
Develop Security Documentation									0	
Conduct Testing	1-2	4	0-1	2-5	4	3	5	3	22-27	agile compatible

Figure 5.6: Results of security activity analysis of NIST 800-64 in regard to how agile they are. [Image was taken by the author of this document]

Agility Feature/ Security Activity	Simplicity	Free of Modeling and Documentation	Change Tolerate	Speed of Execution	Informality	People Orientation	Iterative / Divideable	Flexibility	Agility Degree	Evaluation	Cost Effectivity	Benefit
<i>According Case Study</i>												
Security Requirements Analysis	2	0	3	4	5	4	5	4	27	agile friendly	1	1
Agree on Definitions	2	0	3	2	5	4	4	1	21	agile unfriendly	-1	-1
Risk Analyses	2	0	3	2	3	4	5	1	20	agile unfriendly	-1	0
Critical Assets	3	0	4	4	4	4	5	1	25	agile compatible	1	-1
Identify Threats and Develop Artifacts	1	0	2	3	4	1	4	3	18	agile unfriendly	-1	0
Requirement Inspection	2	3	3	3	4	2	5	0	22	agile compatible	1	0
Repository Improvement	3	0	5	5	3	1	5	3	25	agile compatible	1	0

Figure 5.7: Results of security activity analysis of SREP in regard to how agile they are. [Image was taken by the author of this document]

5.4 Assign security activities to SSE-CMM's PAs

5.4.1 Summary of the evaluation results

According to the above presented results, the following activities are the most agile ones and marked as "agile-friendly":

1. Security Requirements Analysis (Cigital Touchpoint, SREP, Microsoft SDL),
2. Role Matrix (Microsoft SDL),
3. Static Analysis (Cigital Touchpoint, Microsoft SDL),
4. Dynamic Analysis (Microsoft SDL),
5. Code Review (Microsoft SDL),
6. Initiate Security Planning (NIST 800-64), and
7. Conduct testing (NIST 800-64).

Activities which we marked as "agile-compatible" are:

1. Attack Surface Analysis/Reduction (Microsoft SDL),
2. Abuse Cases (Cigital Touchpoint),
3. Penetration Testing (Cigital Touchpoint),
4. Critical Assets (SREP),
5. Requirement Inspection (SREP),
6. Repository Improvement (SREP),
7. Categorise Information System (NIST 800-64), and
8. Assess Privacy Impact (NIST 800-64).

We took these *security activities* from multiple SE process and therefore there is redundancy among these security activities. Some of these security activities have the same goal, such as the defining of security requirements. We combined security activities that are very similar in the two listings above. For example, Cigital Touchpoint, SREP, and Microsoft SDL defined a "Security Requirement Analysis". We interpret these security activities as one security activity.

Discussion of results We think that each “agile-friendly” and “agile-compatible” security activity does fit into agile development except of one — SREP’s “Requirement Inspection” activity. We believe that it is not cost-effective to review all documents produced in all security activities performed and to ensure with SSE-CMM, CC assurance requirements and EALs of Common Criteria whether all defined security requirements take all identified threats into account, and that all security objectives are satisfied.

5.4.2 Assign security activities to SSE-CMM’s PAs

The goal of this thesis is to create a secure Scrum process that fulfils the goals of each PA of the SSE-CMM except PA 01 and PA 08, because these PAs only focus on the maintenance of the software and staffing. To achieve this goal, we assign each "agile-friendly" and "agile-compatible" security activity to a PA of the SSE-CMM and thus we show which security activity takes which PA into account. In Table 5.1 we illustrate which PAs of the SSE-CMM are taken into account by these "agile-friendly" and "agile-compatible" security activities. We assigned them ourselves based on our opinion.

Table 5.1 summarizes the results of the analysis performed in Chapter 4 and 5. This table lists every Process Area of the SSE-CMM and outlines possible ways of how the goals of a PA defined can be achieved. For this reason we took the solutions identified in Chapter 4 and we further suggest all security activities evaluated in this Chapter, which we marked as "agile-compatible" or "agile-friendly". The "Process Area" column describes the PA of SSE-CMM, whereas the "Does standard Scrum support a specific PA?" column has the purpose to check whether Scrum supports this PA. In this column we also suggest possible ways of how Scrum can handle a PA. In the "Security activities that focus on a Process Area" we illustrate which security activities analysed in this Chapter take a particular PA into account.

Process Area (PA)	Does standard Scrum support a specific PA?	Security activities that focus on a Process Area
PA 01: Administer Security Controls	Not considered.	
PA 02: Assess Impact	Scrum does not support this PA.	<ul style="list-style-type: none"> • SREPS’s “Critical Assets” activity, • Microsoft SDL’s “Role Matrix” activity, • NIST 800-64’s “Categorise Information System activity”, and • NIST 800-64’s “Assess Privacy Impact” activity.
PA 03: Assess Security Risk	Scrum does not support this PA.	Microsoft’s DREAD method (is part of Microsoft’s Threat Modeling method).
PA 04: Assess Threat	Scrum does not support this PA.	Cigital Touchpoint’s “Abuse Cases” activity.

PA 05: Assess Vulnerability	Scrum does not support this PA.	<ul style="list-style-type: none"> • Microsoft SDL's Dynamic Analysis activity, • Static Analysis (is a security activity of Cigital Touchpoint and Microsoft SDL), and • Microsoft SDL's Attack Surface Analysis/Reduction activity.
PA 06: Build Assurance Argument	<p>Scrum does not support this PA.</p> <p>Scrum does not specify that for each software increment security requirements have to be defined. To ensure that the software meets the expectations of the customer in regard of security, the customer and a security expert can be involved in defining security requirements and acceptance criteria of a software increment. With assurance methods of PA11 it has to be assured that all security requirements defined are met.</p>	Assurance methods of PA11.
PA 07: Coordinate Security	<p>Scrum supports this PA.</p> <p>Security can be coordinated via small team communication, self-organized teams, Daily Standup meetings, Product Backlog Refinement meetings, Sprint Planning meetings, and Sprint Review meetings.</p>	-
PA 08: Monitor Security Posture	Not considered.	-
PA 09: Provide Security Input	<p>Scrum partially supports this PA.</p> <p>If developers have security expertise, security considerations can be included in user story refinements and in code reviews. Developers can use SAFECode's security-related user stories to define security requirements.</p>	SREP's "Repository Improvement" activity.

<p>PA 10: Specify Security Needs</p>	<p>Scrum partially supports this PA. To define security requirements of a user story so-called security-related user stories can be defined by Product Owner, customer and developers and defined as acceptance criteria for the corresponding functional user story. SAFECode’s security related user story templates should be used to split security-related user stories into estimable tasks. Furthermore, a security expert should be added in the refinement of user stories and Sprint Planning meetings to help the developers to perform security engineering activities, such as risk assessments.</p>	<ul style="list-style-type: none"> • A "Security Requirement Analysis" (is an security activity of Cigital Touchpoint, SREP, and Microsoft SDL), and • NIST 800-64’s "Initiate Security Planning" activity.
<p>PA 11: Verify and Validate Security</p>	<p>Scrum partially supports this PA. According to Scrum, testing and code reviews should be performed. In code reviews security flaws in the source code can be identified by manually reviewing it. By writing security tests it can be verified whether the source code meets the defined security requirements. Security testing methods which can, from our point of view, be used are: Web vulnerability scanning, and whitebox testing, such as unit- and system testing</p>	<ul style="list-style-type: none"> • Cigital Touchpoint’s "Penetration Testing" activity, • Static Analysis (is a security activity of Cigital Touchpoint and Microsoft SDL), • Microsoft SDL’s "Dynamic Analysis" activity, • Microsoft SDL’s "Code Review" activity, and • NIST 800-64’s "Conduct Testing" activity.

Table 5.1: How Scrum handles SSE-CMM’s PAs and which security activities analysed can be used to take a Process Area into account.

5.4.3 Integrating security activities into Scrum

Table 5.1 lists for each PA one or more security activities that can be integrated into Scrum. Our goal is to integrate several of them into Scrum to create a secure Scrum process that takes all PAs of the SSE-CMM into account without losing too much agility. In the next paragraphs we discuss which security activities we integrate into Scrum. We discuss for each PA that Scrum does not or does only partially support, which security activities we integrate into Scrum in order that Scrum takes a particular PA into account. In Chapter 7 we integrate the suggestions below into Scrum and propose a whole secure Scrum process and verify it’s provided agility and security in Chapter 8.

5.4.3.1 PA 02–05

PA 02 ("Assess Impact"), PA 03 ("Assess Security Risk"), PA 04 ("Assess Threat"), and PA 05 ("Assess Vulnerability") are related to each other and can be combined in a threat modeling- or a risk analysis

method. However, each risk analysis- and threat modeling methodology analysed is agile-unfriendly. Therefore, an agile-friendly risk analysis/threat modeling method would be of interest. Our approach is to create with several agile-friendly and agile-compatible security activities that take PA 02–05 into account an agile-friendly risk analysis method. In Chapter 6 we combine several of the security activities from Table 5.1 that take PA 02 ("Assess Impact"), PA 03 ("Assess Security"), PA 04 ("Assess Threat"), and PA 05 ("Assess Vulnerability") into account and combine these security activities into an agile risk analysis methodology and integrate this methodology in Chapter 7 into Scrum.

5.4.3.2 PA 06

We discussed in Section 4.1.3.1 an agile-friendly way of how PA 06 can be taken into account. In summary, to fulfil the goal to generate security requirements, deriving from the goals which prove to the customer that the software being developed is actually secure, we suggest to conduct the following three steps:

1. A security specialist should identify security policies, standards and compliance regulations and should derive requirements from them with the help of the customer. The customer should be involved in defining all high and low-level requirements, corresponding to acceptance criteria and test procedures.
2. Further, requirements should be derived from threats identified from our agile risk analysis method, which we describe in Chapter 6, and be prioritized with the help of the customer.
3. As a third step the security requirements defined should be validated with different testing- methods and techniques, which we describe in the "PA 11" paragraph in this section.

By applying the first- and second point, the security requirements reflect the customer's expectations of the software. The involvement of the customer is agile-friendly, as stated in the agile manifesto. By performing the third point it is assured that the security requirements defined are met.

5.4.3.3 PA 09

The aim of PA 09 is to design and implement security architectures, and specific implementation alternatives to meet the security requirements defined. No security activity analysed that fulfils the goal of PA 09 is agile-friendly or agile-compatible. For example, Microsoft SDL's "Design Requirements" activity fulfils this PA, because the aim of this activity is to create a secure architecture, but this activity is not agile-friendly or agile-compatible. In the next two paragraphs we discuss a possible way of how Scrum can cover PA 09.

NIST-800-64's "Select and document security controls" activity To fulfil the goal of PA 09 we adopt the main idea of NIST 800-64's "Select and Document Security Controls" activity [47, p. 23–24] that one selects a security control, tailors it and then document it (we described NIST 800-64's "Select and Document Security Controls" activity in Section 5.2). However, we do not adopt the specific steps to select them from the in NIST SP 800-53A [66] provided security catalogue, to tailor and to document them as stated by NIST 800-64. We suggest that the developers select one of the in our agile risk analysis' "Mitigation Strategies" step identified security controls (We explain this agile risk analysis method in Chapter 6 and we explain this specific step in Section 6.9). The developers should select these security controls, tailor them in such a way that these security controls fit for a specific user story, and implement these security controls. In this way, PA 09's goal to identify and implement solutions to meet security requirements defined is fulfilled. We name this new security activity "Document Security Controls".

SREP's "Repository Improvement" activity We further integrate SREP's "Repository Improvement" into Scrum, because by integrating this activity into Scrum, the identification of security controls to meet security requirements defined can be speeded up. In the "Repository Improvement" activity will be discussed which threats, requirements (model elements) can occur in the future, which will then be documented to speed up further analyses [52].

In our opinion, SREP's "Repository Improvement" activity would fit well into Scrum. Scrum has a meeting, the so called Sprint Retrospective meeting, which focuses on improving the performance in work. In this meeting all team member discuss what they should continue to do, stop doing, or what they should start to do to increase the performance or quality of their work [88]. Obviously, documenting implemented security-related user stories to support further analyses in regard to security can improve the performance of work and would therefore provide much benefit. Hence, we adopt the idea of SREP's "Repository Improvement" and suggest to document all security-related user stories which are likely to occur on, for example, a Twiki page so that that they can be queried fast in further analyses.

5.4.3.4 PA 10

The goal of PA 10 is to specify the security needs of the system. To specify security needs we integrate NIST 800-64's "Initiate Security Planning" activity into Scrum. In NIST 800-64's "Initiate Security Planning" activity the company discusses with stake holders of the project about which security policies should be established, and which high-level security requirements should be taken into account. We also adopt the idea of Cigital Touchpoint's, SREP's, Microsoft SDL's, and NIST 800-64's "Security Requirement Analysis" to specify security requirements by deriving them from functional requirements, laws, standards and security policies. By integrating these security activities into Scrum PA 10's goal that all stake holders of a software project are aware of the security needs of the system is fulfilled.

5.4.3.5 PA 11

As can be seen in the PA 11 row of Table 5.1, several agile-friendly methods, techniques and tools exist that can be used to ensure that the software developed meets the security requirements defined. We discuss in the next paragraphs which set of security activities listed in the PA 11 row of Table 5.1 can be used to fulfil our goal to verify and validate whether the software developed is secure against the OWASP Top 10.

NIST 800-64's "Conduct Testing" activity NIST 800-64's "Conduct Testing" activity does not state which testing- techniques, methods, or tools should be used to ensure that the software is really what the customer wants in regard of security. NIST 800-64 defined only that testing should be automated as much as possible. In our opinion, it is important that testing is as time-efficient as possible to fit into agile development and we therefore agree that testing should be automated as much as possible. From our point of view, developers should write as many unit, integration and system tests as possible to verify that the software is really what the customer wants in regard of security. However, with this kind of testing only functional security requirement testing can be performed. To test whether the software implemented meets non-functional security requirements defined code reviews, static code analysis, or penetration testing can be used.

Microsoft SDL's "Code Review" activity, Cigital Touchpoint's and Microsoft SDL's "Static Analysis" activity, and Microsoft SDL's "Dynamic Analysis" activity In Scrum, code reviews should be regularly conducted [87]. Hence, Microsoft SDL's "Code Review" activity would fit well in Scrum, because Microsoft SDL's "Code Review" activity is a kind of code review. In this activity one reviews whether the source code contains security flaws. To help the reviewer to identify as much security flaws as possible static- and dynamic code analysis tools can be used. Static code analysis tools, such as

Checkmarx's Static Code Analysis tool (SAST) [22] are capable of testing whether the source code is vulnerable to the OWASP Top 10. In our opinion, these tools are time-efficient and deliver quick and reliable results and speed up the manual code review. We think that especially semi-automatic code reviews from a security expert with support of static code analysis tools is efficient in regard of time, because in the context of a single user story it does not take long to analyse the implementation.

Additionally, we recommend performing the XP practice pair programming for high-security critical software components. One developer should write the code, whereby a security specialist looks for coding failures or even security flaws in the code. In this way, security flaws can be mitigated early.

Penetration Testing From our point of view, by integrating the following security activities into Scrum: NIST 800-64's "Conduct Testing", Microsoft SDL's "Code Review", Cigital Touchpoint's and Microsoft SDL's "Static Analysis", and Microsoft SDL's "Dynamic Analysis", our goal to verify and validate whether the software developed is secure against the OWASP Top 10 is met.

However, we recommend that additionally a pentester conducts manual penetration tests regularly, but not every Sprint, because conducting it for every user story would result in too much testing effort. By conducting penetration testing a security expert ensures that the software is not "hackable", which provides in our opinion, high evidence that the software is secure. However, penetration testing requires a security expert, which may not always be available in practice. If no security expert is available, we argue that especially automatic "all-in-one" penetration testing- or web vulnerability scanning tools (see Section 3.7.2 for greater details) should be integrated into agile development. These tools deliver quick and trustworthy results and they do not require a security expert [78, p. 16]. Hence, automatic penetration testing or web vulnerability scanning can fit well into agile development.

Web vulnerability scanning Conducting web vulnerability scanning is a time-efficient way to identify software flaws in a piece of software [78, p. 16]. Nevertheless, in contrast to manual penetration testing, performing only web vulnerability scanning is not sufficient to ensure whether the software is secure, due to the fact that it is only a black-box testing method. However, by conducting web vulnerability scanning one receives a quick overview of whether the software developed contains security flaws. We recommend that a Quality Engineer and a developer who developed a certain user story test with black box web vulnerability scanner, whether the code that the developer implemented contains security flaws. If possible, only the new implemented functionality should be tested with this tool. For example, if a new web site was implemented, only this site should be tested with the tool. In the next step the results of this analysis are triaged whether it contains false positives. If this is the case, the Quality Engineer and the developer write a documentation, which lists all false positives that occurred including an explanation why they are no security flaws. We call this new security activity "Pair Penetration Testing".

Summary of PA 11

We think that there is no single ideal security testing approach and that security assurance should be achieved by several security activities instead of conducting only one technique or method. Developers should write as many unit, integration and system tests as possible. Code reviews should be performed for each user story with the help of static- and dynamic code analysis tools to ensure that all security flaws in the source code are identified. Next, pair penetration testing should be used for each Sprint to scan for vulnerabilities in the code and a pentester should perform manual penetration tests regularly, but not every Sprint. In this way, our goal to verify and validate whether the software developed is secure against the OWASP Top 10 is fulfilled.

Chapter 6

Agile Risk Analysis

6.1 Motivation

As can be seen in Chapter 5, existing risk analysis and threat modeling methodologies do not fit into agile development. Nevertheless, it is essential that some kind of risk analysis/threat modeling is performed. Hence, we decided to create our own methodology which is based on the ideas of the agile friendly activities evaluated in the previous Chapter which we listed in Table 5.1 as possibilities to take SSE-CMM's PA 02 ("Assess Impact"), PA 03 ("Assess Security Risk"), PA 04 (Assess Threat), and PA 05 ("Assess Vulnerability") into account. To be more specific, we take the following activities: "Abuse cases" (see Section 5.2.2.2), "Role Matrix" (see Section 5.2.1.2), "Critical Assets" (see Section 5.2.3.4), "Attack Surface Analysis/Reduction" (see Section 5.2.1.6), and "Categorize Information System" (see Section 5.2.4.2) and combine them into one methodology.

6.2 Intent of this chapter

This Chapter aims to describe an agile risk analysis on a high-level. This agile risk analysis is part of the secure Scrum process that we describe in Chapter 7. We call this methodology agile risk analysis (ARA), because it is based on the most agile-friendly security activities analysed in the previous Chapter. Our approach is to take the steps of OWASP's Application Threat Modeling (OATM) (see Section 3.4.6 for greater details) as reference, remove activities from their methodology which are not necessary in our opinion, modify some to fit into agile, and replace steps which have the same goal as some of the agile activities which we listed in Section 6.1. We assume for this Chapter that the reader is familiar with OATM and the in Section 6.1 listed agile friendly security activities. Table 6.1 illustrates which activities of OATM stay the same, which are removed and which are replaced with one of the above described agile friendly ones. The first column indicates the original name of the step according to OWASP Application Threat Modeling, whereas the second states our new name of this step. The last column describes if the step is removed, is modified or stays the same. We do not refer to it as Threat Modeling, because it does not produce a threat model. The aim of our methodology is to identify threats which can occur if a user story is implemented. Furthermore, the identified threats are prioritized and mitigation strategies to tackle them are determined.

OATM's original name	Our name	Our modification of this step
External Dependencies	Determine External Dependencies and make Security Assumptions	We adopt this step and further recommend to use dependency checker tools.
Assets	Identify assets	We slightly modify this step by using a database scheme instead of using a table. In order to identify assets we adopt the idea of Cigital Touchpoint's "Critical Asset" activity. Furthermore, we add NIST 800-64's "Categorize Information System" activity for analysing the impact on an asset.
Trust Levels	Identify trust levels (Determine users of the system)	We adopt and combine the idea of OATM's Asset Table and the idea of Microsoft SDL's "Role Matrix" and integrate it into our database scheme mentioned above.
Entry Points	Determine Entry Points	We adopt Microsoft's Attack Surface Analysis/Reduction activity in order to determine entry points. To document them we add the idea of OATM's Entry Point table into our database scheme.
Data Flow Diagrams	-	We remove this step.
Threat Analysis	Identify Threats	Instead of using STRIDE or ASF, this step is based on the idea of Cigital Touchpoint's "Abuse Cases" activity and SAFECODE's security-related user story templates.
Ranking of Threats	Ranking of Threats with DREAD	Stays the same.
Countermeasure Identification and Mitigations Strategies	Mitigations Strategies	Is slightly modified.

Table 6.1: Agile risk analysis high-level description.

Figure 6.1 illustrates all high-level points of our new agile risk analysis (ARA) that we mentioned in Table 6.1. In the next few sections we describe all steps of Figure 6.1 in greater detail.

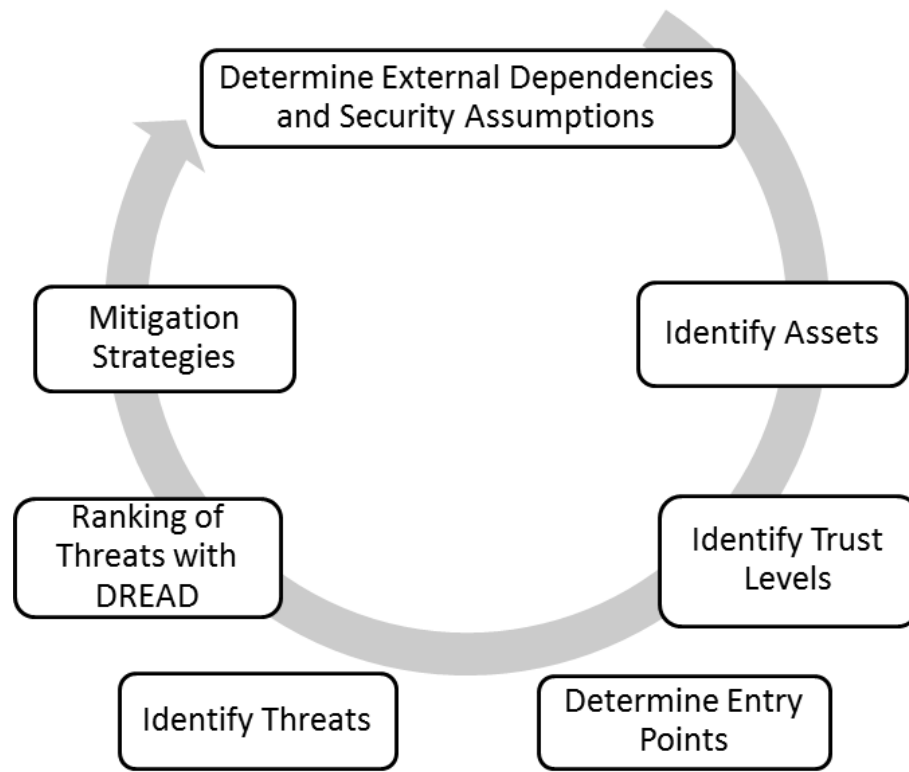


Figure 6.1: Our agile risk analysis (ARA).

6.3 Determine External Dependencies and Security Assumption (External Dependencies)

Discussion of OATM's step

We adopt this step from OATM, because we think that it is essential to define in which environment the software will be deployed and on which security assumptions further steps should be taken. For example, that sensitive data is only sent via TLS or other secure channels, and that the web server restricts access to the filesystem with the help of operating system access control lists (ACLs). These lists specify which users are allowed to read, write or execute a particular file on the server. These assumptions help to determine which security controls are already in place and which would thus be a waste of time to implement. This can speed up further analyses. However, it is important to mention that further steps rely on this analysis, hence, it should be performed thoroughly. If, for example, no secure connection is used for sensitive data communication, the developers have to encrypt the data on their own.

When this step should be performed

From our point of view, these steps should be performed at the beginning of each agile project and be adopted if external dependencies or security assumptions change, and do not have to be performed for each user story, because the system on which the software is deployed is not changed frequently.

Our modification of the step

We recommend to attach a link of user stories that rely on security assumptions and vice versa. In this way, if the assumptions change one knows which user stories have to be revisited.

Additionally, we add a further step where software dependency checker tools are executed every Sprint in order to find vulnerabilities in libraries that are used in a project. One example would be the OWASP Dependency Check [80]. The aim of this tool is to scan the source code for vulnerable libraries and to display them. This tool can be run on an automation server, like Jenkins [71]. The development team can then frequently triage the results. If they find a vulnerable library, they should document if this vulnerability has an effect on their software. If this is the case, they should document it as a threat and handle it in the "Mitigation Strategies" step of this methodology.

6.4 Identify Assets

Discussion of OATM's step

According to OATM, in this step all assets of a piece of software and which users have access to it should be listed in a so called Asset Table. However, OWASP never stated how assets can be determined.

Our modification of the step

A possible way of how assets can be identified and prioritized In order to identify as much assets as possible, we reckon that the analysts should do interviews with asset owners and stakeholders, or they should derive assets from functional requirements as stated by SREP's "Critical Assets" activity (see Section 5.2.3.4 for greater detail). As next step, in our opinion, it is important to rate the importance of the assets, because with this information one can determine whether threats which aim to obtain, alter, or deny access to an asset are relevant. In order to achieve this, we adopt the idea of NIST 800-64's "Categorize Information System" activity (see Section 5.2.4 for greater detail) in which the level of impact of each asset on the security objectives confidentiality, integrity and availability is assessed. Stine et al. [92] provides a metric which can be used to perform an impact assessment. In Figure 6.2 this metric can be seen. It illustrates how the impact on availability, integrity and confidentiality can be categorized into low, medium, and high. In summary, we are now able to identify assets, to list them, and to rate their impact levels.

A possible way of how assets can be documented However, in big projects these tables where the assets are listed can grow very fast and are not maintainable. Our extension is that a general database is used in which all assets and information systems of an agile project are stored, instead of using the tables which OATM suggests. We recommend that if new assets are introduced in a user story, these assets are rated according to the Metric which can be seen in Table 6.2 and stored in this database. Furthermore, a link to the user story which implemented this functionality is attached to a database entry and in the user story a link to this database entry is added. In few words, every user story that impacts assets is added as a link to database entries and vice versa. In this way, assets are added iteratively to the project and one knows which user stories impact which assets. Thus, if the impact level of an asset is changed, one knows which user stories must be revisited.

SECURITY OBJECTIVE	POTENTIAL IMPACT		
	LOW	MODERATE	HIGH
Confidentiality Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information. [44 U.S.C., Sec. 3542]	The unauthorized disclosure of information could be expected to have a limited adverse effect on organizational operations, organizational assets, or individuals.	The unauthorized disclosure of information could be expected to have a serious adverse effect on organizational operations, organizational assets, or individuals.	The unauthorized disclosure of information could be expected to have a severe or catastrophic adverse effect on organizational operations, organizational assets, or individuals.
Integrity Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity. [44 U.S.C., Sec. 3542]	The unauthorized modification or destruction of information could be expected to have a limited adverse effect on organizational operations, organizational assets, or individuals.	The unauthorized modification or destruction of information could be expected to have a serious adverse effect on organizational operations, organizational assets, or individuals.	The unauthorized modification or destruction of information could be expected to have a severe or catastrophic adverse effect on organizational operations, organizational assets, or individuals.
Availability Ensuring timely and reliable access to and use of information. [44 U.S.C., Sec. 3542]	The disruption of access to or use of information or an information system could be expected to have a limited adverse effect on organizational operations, organizational assets, or individuals.	The disruption of access to or use of information or an information system could be expected to have a serious adverse effect on organizational operations, organizational assets, or individuals.	The disruption of access to or use of information or an information system could be expected to have a severe or catastrophic adverse effect on organizational operations, organizational assets, or individuals.

Figure 6.2: NIST SP 800-60's metric for categorizing the impact of information systems on confidentiality, availability, and integrated. Adopted from [92].

In the next section we explain our database scheme in greater detail. In further steps of this methodology we add additional data to the database scheme and present the extensions. In the "Determine Entry Points" step of this methodology, we display the final database scheme.

First iteration of our database scheme

Figure 6.3 illustrates the first iteration of our database scheme. The Figure is based on aspects of a Database Structure Diagram. Our database scheme consists of three tables. The first shows a User-Story Table where the user stories are listed, whereas the second presents the assets of the system and their impact levels. In the last table (RiskAnalysis Table) it is shown which user story implemented a specific asset. Figure 6.4 illustrates our database scheme with an example. The example shows how such a database scheme looks like on an minimalistic internal hospital software. The system can display data of patients and can show an operation timetable. Furthermore, it includes a timestamp system in which every user has to login when they come to work and has to log out when they go home from work. It also includes an admin site.

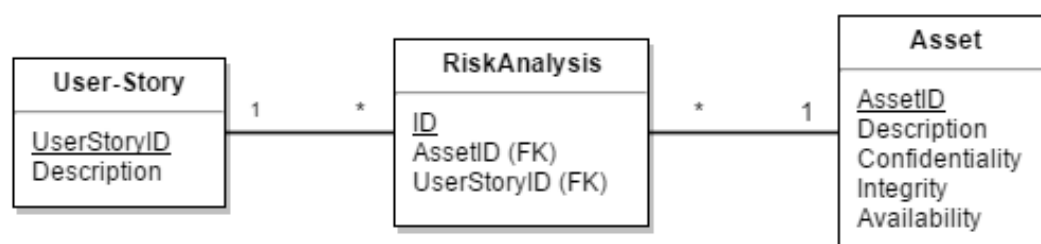


Figure 6.3: Database scheme of our agile risk analysis method (iteration 1).

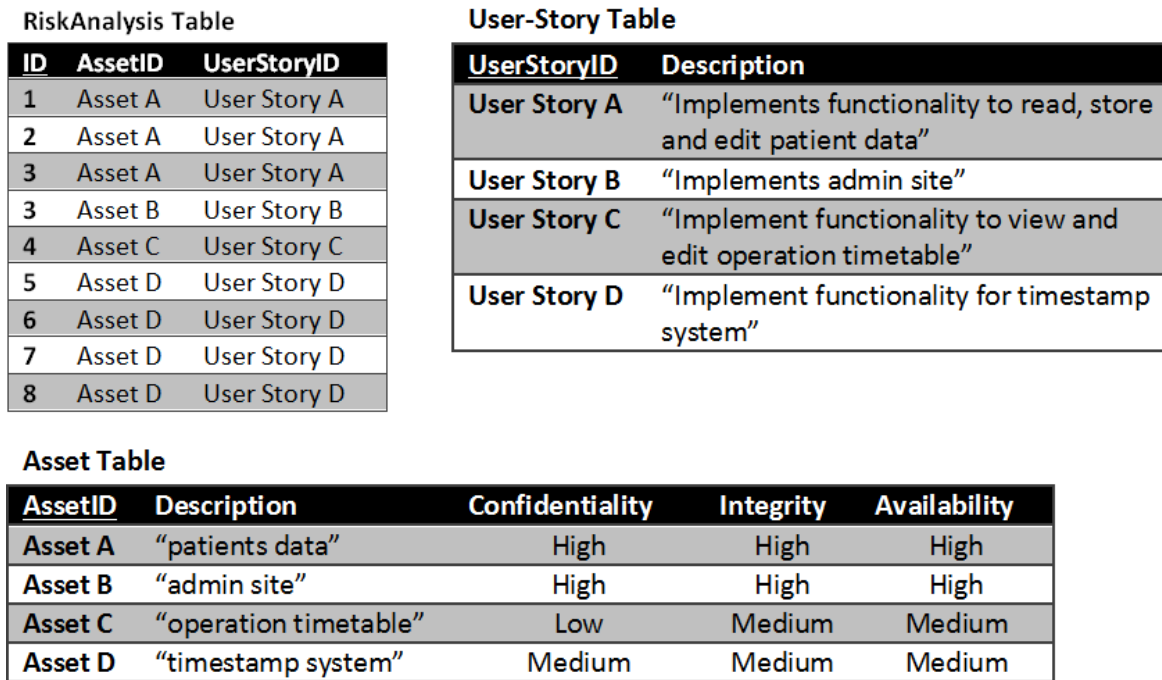


Figure 6.4: Example of our final database scheme (iteration 1).

6.5 Identify Trust Levels

Discussion of OATM's step

OATM claims that a table should be used to list all trust levels of the system. A trust level can be a user of the system or an attacker. This table should include the id of a user, his name and a description.

Our modification of the step (First extension of our database scheme)

We adopt their approach and extend our database scheme by adding OATM's table in our database scheme. We also adopt the idea of Microsoft SDL's "Role Matrix" activity in which for each asset is illustrated which user is allowed to create, read, write or execute it. We expand the "RiskAnalysis Table" of our database scheme with this metadata and extend it in such a way that it includes all users which have read, write or execute rights to an asset. Additionally, the "RiskAnalysis Table" should hold a link to all user stories which implemented the functionality that allows the user to access an asset. This data should be added in each user story. The metadata for access rights is usually only needed for files. However, an asset can also be the ability to delete users from the system, a login session, or the access right to a specific site. To clarify why it makes sense to add this metadata for all kinds of asset, we present some examples. For the first example listed above some kind of procedure has to be called in order to remove a user. In this case, the asset is this procedure. A user has to have the access right "execute" in order to be allowed to execute it. In case of the second and third example, a user can only access a specific site or a login session if the "read" value is set to true.

Second iteration of our database scheme

Figure 6.5 shows the discussed extensions to the database scheme and Figure 6.6 illustrates our database by the means of the internal hospital software. We added a "User Table" to our database scheme and we extended the "RiskAnalysis Table" with the "read", "write", and "execute" rights which a user has on an asset.

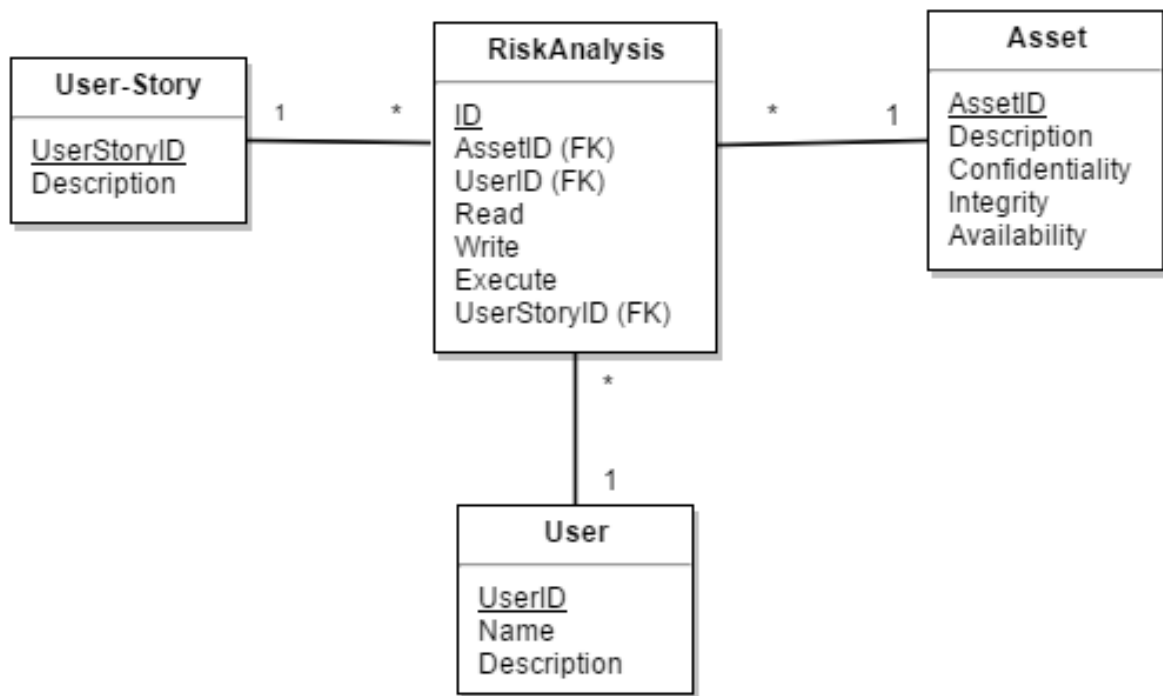


Figure 6.5: Database scheme of our agile risk analysis method (iteration 2).

RiskAnalysis Table

ID	AssetID	UserID	Read	Write	Execute	UserStoryID
1	Asset A	User A	TRUE	TRUE	FALSE	User Story A
2	Asset A	User B	TRUE	FALSE	FALSE	User Story A
3	Asset A	User C	TRUE	TRUE	FALSE	User Story A
3	Asset B	User B	TRUE	FALSE	FALSE	User Story B
4	Asset C	User A	TRUE	TRUE	FALSE	User Story C
5	Asset D	User A	TRUE	TRUE	FALSE	User Story D
6	Asset D	User B	TRUE	TRUE	FALSE	User Story D
7	Asset D	User C	TRUE	TRUE	FALSE	User Story D
8	Asset D	User D	TRUE	TRUE	FALSE	User Story D

User-Story Table

UserStoryID	Description
User Story A	"Implements functionality to read, store and edit patient data"
User Story B	"Implements admin site"
User Story C	"Implement functionality to view and edit operation timetable"
User Story D	"Implement functionality for timestamp system"

Asset Table

AssetID	Description	Confidentiality	Integrity	Availability
Asset A	"patients data"	High	High	High
Asset B	"admin site"	High	High	High
Asset C	"operation timetable"	Low	Medium	Medium
Asset D	"timestamp system"	Medium	Medium	Medium

User Table

UserID	Name	Description
User A	"doctor"	"..."
User B	"administrator"	"..."
User C	"nurse"	"..."
User D	"cleaner"	"..."

Figure 6.6: Example of our final database scheme (iteration 2).

6.6 Determine entry points

Discussion of OATM's step

In this step OATM aims that entry points where attackers can start their attacks to gain access to an asset should be identified. An entry point, can be, for example, a login page or a simple html form. To record them, OATM claims to use a table. Each entry point and who has access to is listed in this table. The entry point table of OATM consists of four columns. The first represents the id of an entry point. The second displays its name and the third gives a description of the entry point. The last displays a list of trust levels, which have access to this entry point. However, this table does not display from which entry point one can interact to a specific asset. OATM did not explain how these entry points can be identified.

Discussion of our modifications of the step (Second extension of our database scheme)

We adopt OATMs idea of a table for entry points and add an "Entry Point Table" to our database scheme. Our table consists of the first three columns of OATMs table. We further add an additional column to our "RiskAnalysis Table". This column contains the id of an asset. In this way, we are in contrast to OATMs table able to illustrate through which entry point a user has access to a particular asset.

When a database entry should be added For each functionality which is implemented in a user story that allows user to access a particular asset through a specific entry point an entry in our "RiskAnalysis Table" should be added. This database entry consists of the id of the asset, the id of the entry point through which the asset is accessible, and the id of the user story in which this functionality was implemented. Furthermore, this entry point contains the id of the user who has read, write, and/or execute rights on this asset.

Advantages This information can speed up the attack surface analysis, due to the fact that the analysts can quickly look up through which entry point a user has access to a specific asset. If additional entry points are developed or they are modified in further user stories, the database has to be updated. The goal of this idea is that the analysts are able to keep track the changes which user stories may cause on assets and on access restrictions and to have a documentation of it.

A possible way of how entry points can be identified We think that an Attack Surface Analysis [34, p. 78–90] (see Section 5.2.1 for a detailed description) should be used to determine them. The aim of an attack surface analysis is to identify the entry points of the software where an attacker can start their attack [34]. For achieving this, the analysts can evaluate the software design, or they can derive them from functional requirements. After they identified each entry point, they should ask themselves whether this functionality is really needed or if a specific user should really have access to a particular asset.

Third iteration of our database scheme

In Figure 6.7 we display the final database scheme and Figure 6.8 shows how our changes to the database scheme looks like with an example. The software in the example has four users, which can be seen on the "User Table", four assets ("Asset Table"), which were implemented in four user stories ("User-Story Table"). The "Asset Table" contains an id, a description and in the last three columns the level of impact on confidentiality, integrity and availability is shown. The "RiskAnalysis Table" illustrates the access rights of each user and which user story implemented a specific access right. The first column highlights the unique identifier of the table entry, whereas the second column presents the

id of an asset. The third, fourth and fifth column show which user has reading, writing or executing rights on an assets. The sixth column states the id of the user story which implemented this functionality and the last column shows from which entry point an asset is accessible. In this example you can see in the second row of the "RiskAnalysis Table" that the admin user has reading access to the patient data. That can be, for example, a security flaw. In this case, the analysts should ask themselves whether it is absolutely necessary that the admin user has access to patient data. If not, a threat has been found and should be handled taking the steps described in "Mitigation Strategies".

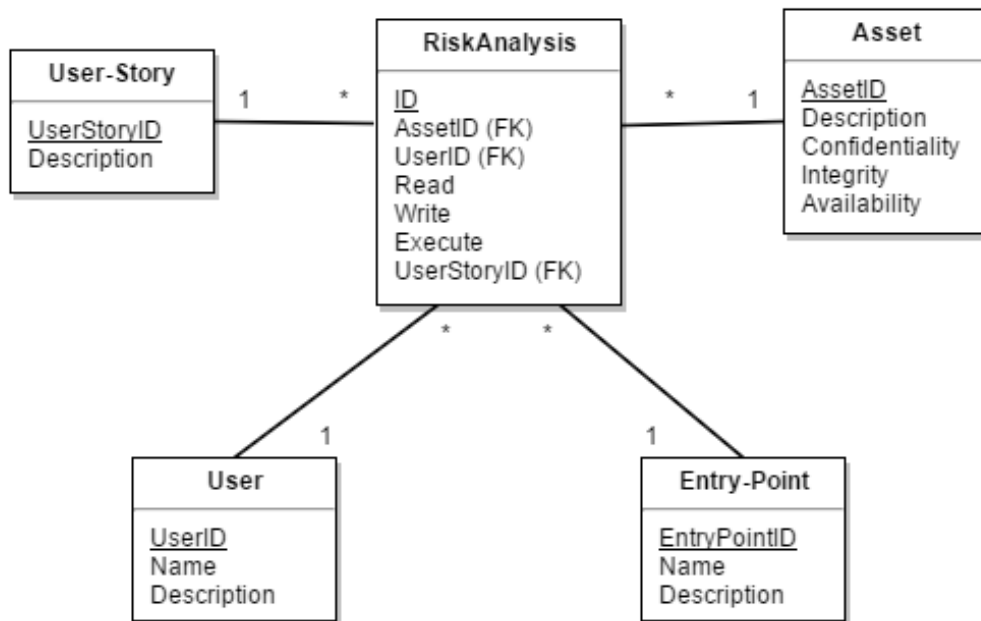


Figure 6.7: Final database scheme of our agile risk analysis method.

RiskAnalysis Table

ID	AssetID	Read	Write	Execute	UserStoryID	EntryPointID
1	Asset A	User A	TRUE	TRUE	FALSE	Entry Point A
2	Asset A	User B	TRUE	FALSE	FALSE	Entry Point A
3	Asset A	User C	TRUE	TRUE	FALSE	Entry Point A
4	Asset B	User B	TRUE	FALSE	FALSE	Entry Point B
5	Asset C	User A	TRUE	TRUE	FALSE	Entry Point B
6	Asset D	User A	TRUE	TRUE	FALSE	Entry Point C
7	Asset D	User B	TRUE	TRUE	FALSE	Entry Point C
8	Asset D	User C	TRUE	TRUE	FALSE	Entry Point C
8	Asset D	User D	TRUE	TRUE	FALSE	Entry Point C

Asset Table

AssetID	Description	Confidentiality	Integrity	Availability
Asset A	"patients data"	High	High	High
Asset B	"admin site"	High	High	High
Asset C	"operation timetable"	Low	Medium	Medium
Asset D	"timestamp system"	Medium	Medium	Medium

Entry Point Table

EntryPointID	Name	Description
Entry Point A	"Entry Point A"	"Search site for patients"
Entry Point B	"Entry Point B"	"Central menu site"
Entry Point C	"Entry Point C"	"Login site for timestamp system"

User-Story Table

UserStoryID	Description
User Story A	"Implements functionality to read, store and edit patient data"
User Story B	"Implements admin site"
User Story C	"Implement functionality to view and edit operation timetable"
User Story D	"Implement functionality for timestamp system"

User Table

UserID	Name	Description
User A	"doctor"	"..."
User B	"administrator"	"..."
User C	"nurse"	"..."
User D	"cleaner"	"..."

Figure 6.8: Example of our final database scheme.

We recommend to developing a web interface where each asset or user can be searched. This graphical user interface should be capable of displaying which user has access to an specific asset, from where it is accessible and in which user stories a functionality was implemented that allows accessing it.

6.7 Identify Threats

Discussion of our modifications of the step

We remove the typical decomposing of the software from OWASP's Application Threat Modeling, where data flow diagrams (DFD) are drawn in order to analyse the flow of data within a software and to analyse possible manipulations of attackers on this data. In our opinion, these diagrams are effective to identify threats, nevertheless, they are not change tolerant, and are too time consuming to draw and to maintain in practice. Therefore, we remove it. However, STRIDE can still be used, because it is only a categorization methodology for threats. We think that STRIDE is a great approach and can be used in addition to our method, which we explain in the next paragraph.

Instead of DFDs, we purpose that brain storming sessions should be held, where threats are identified with the help of security-related user story templates from SAFECode [11]. We already discussed them in Section 4.1.2.1, but we quickly summarize them here. These templates were created to support the formalization of security requirements in an agile development process. They defined 36 templates in which they describe which non-functional security requirements can arise when implementing a generic functional user story. Moreover, they illustrated in which tasks these security-related user stories can be divided, which SAFECode fundamental practices [16], such as using anti-cross site scripting libraries, should be used and provide links to one or more CWE entries. A CWE entry describes a common weakness in greater detail and states how likely it is to exploit it, provides code snippets of common coding failures, and illustrates how the functionality can be implemented in a secure way. We provided an example of such a security related user story template in Chapter 4 in Figure 4.1.

These security-related user story templates are very generally defined, but from our point of view, more specific templates can be added over time to obtain a quicker security analysis. We recommend that common specific security-related user story templates are formalized in new templates to speed up the identification of threats.

How security-related user stories should be used to identify threats

We divide our approach of using these security-related user story templates in order to support the identification of threats into four steps.

1. First, the analysts evaluate which security-related user stories are required for a specific functional user story.
2. As a second step, the analysts determine which vulnerabilities can occur if a user story is implemented. To support this process the from SAFECode provided links to CWE entries should be used. Furthermore, results from, for example, previous static code analysis scans, code reviews and penetration tests can also be used to identify common vulnerabilities.
3. In the third step, the analysts formulate security-related user stories into abuse cases, such as "The attacker is able to get access to the asset XY, which is categorized as high security critical".
4. Furthermore, analysts map specific attack scenarios to abuse cases with the help of attacks described in the OWASP Top 10 document. The analysts should ask themselves whether it is possible to attack the system with attacks described in this document and should then formulate specific abuse cases. These abuse cases should be formulated either formally or with the help of diagrams, which can be developed fast and should be attached to the corresponding security-related user story. This means that a security-related user story can contain one or more abuse cases. The goal

is to illustrate which abuse cases are possible if this security-related user story is not taken into account, to identify further threats and to support the writing of specific test cases.

6.8 Ranking of Threats with DREAD

We adopt this step from OATM. After all threats are identified (for example specific abuse cases), risks are evaluated with DREAD (see Section 3.4.3 for greater details). Threats exploit vulnerabilities to get access to an asset. According to DREAD, the risk of a threat depends on the following five points:

- importance of the assets attacked,
- the number of users that use these assets,
- the probability of exploiting vulnerabilities to get access to it,
- the probability of discovering vulnerabilities, and
- the difficulty to reproduce an attack.

Our approach to measure the grades for these five points for specific abuse cases

In order to identify the importance of assets and which users use them the second ("Identify assets") and third step ("Identify trust levels") of this methodology are performed. With the help of the OWASP Top 10 document for particular abuse cases grades for the remaining three points can be measured. The OWASP Top 10 document describes, for each of their Top 10 attacks, grades of the attacker vector, weakness prevalence, weakness detectability and technical impact. Abuse cases describe specific scenarios of how these OWASP Top 10 attacks can be used to abuse an application. Therefore, the grades of the OWASP Top 10 document can be used to measure grades for the remaining three points of DREAD for a specific abuse case. However, these values can greatly vary from application to application. Therefore, the provided grades should only be considered to help rating the abuse cases and the analysts should not see them as carved in stone.

6.9 Mitigation Strategies

In order to handle threats we adopt the "Mitigation Strategies" step of OATM that states that the following mitigations strategies exist: do nothing, remove the feature, turn off the feature, warn the user, implement security controls, or transfer the risk, for example by using contractual agreements. All mitigation strategies, except the "implement security controls" remains the same. Instead of using STRIDE Threat & Mitigation Techniques List or an ASF Threat & Countermeasures List for selecting security controls as stated in OATM, we claim that safeguards of CWE entries, tasks of the security-related user story templates and security controls that are described in the OWASP Top 10 document are used. For example, the security-related user story template from the example above references to [25]. This CWE entry describes that, amongst other issues, JAAS Authorization Framework can be used for implementing this functionality and that Architecture / Design Review Inspection (IEEE 1028 standard) or Correct-By-Construction are high effective to make an assurance argument, but are very costly. They also claim that cheaper ways would be to use Web Application Scanner or Context-configured Source Code Weakness Analyzer, but are not as effective. After some security controls have been selected, they should be documented in the corresponding security-related user story. In order to be more effective we claim that the company should design its own security-related user stories and add them to the already existing ones.

Chapter 7

Proposal for an Agile Secure Process (Secure Scrum)

In this Chapter we first propose a whole secure Scrum process and discuss in Section 7.4 how this process proposal takes the PAs of the SSE-CMM into account. We suggest high level security activities which extend the Scrum development process and explain which Scrum roles are involved and which output of each activity is expected. In order to achieve this we integrate activities and tools which are agile-friendly according to our results in Chapter 5 and we also integrate the steps of our agile risk analysis method that we described in Chapter 6. We integrate all security activities that we discussed in Section 5.4.2. We further integrate all steps of our agile risk analysis process, which we presented in the previous Chapter. In this Chapter we only summarize how each step has to be performed and in each phase refer to the previous Chapter for further details.

In the first Section we illustrate a high level overview of the process. The next Section is dedicated to activities which are performed in the epic and user story definitions and refinements, whereas the next Section outlines quality gates that we use to ensure that all activities have been carried out properly. In Section 7.4 we discuss how this process takes the PAs of the SSE-CMM into account. We assume for this Chapter that the reader is familiar with the ordinary Scrum process, which we explained in Section 3.2.

7.1 High level description

In this Section we discuss first some preliminaries of the process, such as the required security training of the developers, which roles we add to Scrum, and how security requirements are documented in this Secure Scrum proposal. This is then followed by a high-level overview of the process. In order to do so we present two figures, which illustrate this proposal and also highlight the differences to the ordinary Scrum process.

7.1.1 Preliminaries

7.1.1.1 Documentation of Security Requirements

In order to integrate security requirements in an agile process it is necessary to define and document them in some way. In Scrum, epics and user stories are used for defining and documenting functional requirements. In this proposal we use the idea of security-related user stories from SAFECode, which we already discussed in Section 4.1.2.1 to specify security requirements. In a few words, SAFECode defines about 36 general security-related user stories and shows how they can be divided into estimable

tasks. A company should refine them in such a way that they fit for specific functional user stories. Each of them receives a security debt value and should then be added to the Product Backlog. If a team does not develop them in a Sprint, the security debt value of these stories is added to the teams security debt. If this debt exceeds a certain threshold, they must develop these security-related user stories in a separate Sprint. However, as we already mentioned in the above stated Section, from our point of view, it is not necessary to assign a security debt for each story. We claim that high security-critical ones should be added as acceptance criteria for the corresponding functional user story. In this way, it has to be developed in order that the functional user story obtains a "Definition of Done". We also use the idea of [82], which we discussed in Section 2.2, to add a link from every security-related user story to each corresponding functional user story. Thus, one knows which security-related user story is part of which functional user story.

In the next paragraph we summarize how security related user stories are specified in this Secure Scrum proposal. When a technical user story is formalized, one determines which security-related user stories can be needed for this functional user story. As next step, one tailors them to fit for this specific technical user story. All defined security-related user stories are then linked to this functional one and added as Product Backlog item. If it is a high security-critical one, it is added as an acceptance criterion of the technical user story. Whether it is high-security critical is evaluated in our agile risk analysis methodology. If the functional user story has been finished in a Sprint, but the attached security-related user stories have not been completed yet, the security debt value of this story is added to the team's security debt. If this value exceeds a certain predefined value, a security Sprint is performed. All security-related user stories are developed in this security Sprint.

7.1.1.2 Required security training of developers and additional Scrum roles

We require that all developers and Quality Engineers have to be aware of the OWASP Top 10 [77] in order that they can fulfil their functions in this secure software process. Every developer and Quality Engineer should therefore attend security related trainings focusing on these guidelines to get a good understanding of possible attacks and weaknesses.

We add two new roles to this process: a Security Risk Specialist, and a Security Tester. The Security Risk Specialist should assist the developers in performing our agile risk analysis methodology. He should have practical security expertise in performing security risk analyses, such as Microsoft's Threat Modeling. The work of the Security Tester consists of testing the software in regard of security and to help Quality Engineers to perform the "Verification" steps of this proposal. The Security Tester has to know several attack patterns, should be able to write security tests, and to perform penetration tests.

7.1.2 High-level overview of the proposal

In order to illustrate the process we first group the high-level activities into waterfall-like phases, such as requirement, design, implementation and verification. Second, we explain our modifications to the already existing quality gates of Scrum and finally we integrate each phase into Scrum and present our Secure Scrum proposal.

7.1.3 Modifications to the Sprint Retrospective meeting

According to our results in Chapter 5, the Repository Improvement would fit well into Scrum. In the Repository Improvement activity will be discussed which threats, requirements (model elements) can occur in the future, which will then be documented to speed up further analyses [52]. Scrum has a meeting, the so called retrospective meeting, which focuses on improving the performance in work. In this meeting all team member discuss what they should continue to do, stop doing, or what they should start to do to increase the performance or quality of their work [88]. Obviously documenting

implemented security-related user stories to support further analyses in regard to security can improve the performance of work. Hence, we adopt the idea of the Repository Improvement and suggest to document all security-related user stories which are likely to occur on, for example, a Twiki page in order that they can be queried fast in further analyses. We further recommend to archive test-cases used to ensure that the security requirements defined in these security-related user stories are met.

7.1.4 Waterfall process

We adopt the idea of dividing activities into how often they should be performed in a project from [57] as we already mentioned in Chapter 2. We suggest three types: (1) Have to be performed only once per release, (2) Have to be performed once per Sprint, (3) Have only to be performed at the beginning of a project and if major changes in requirements occur. Figure 7.1 outlines all waterfall phases and which security activity is performed in each of them. Furthermore, it also marks the activities according to the numbering explained above. Section 7.3 describes each activity in greater detail.

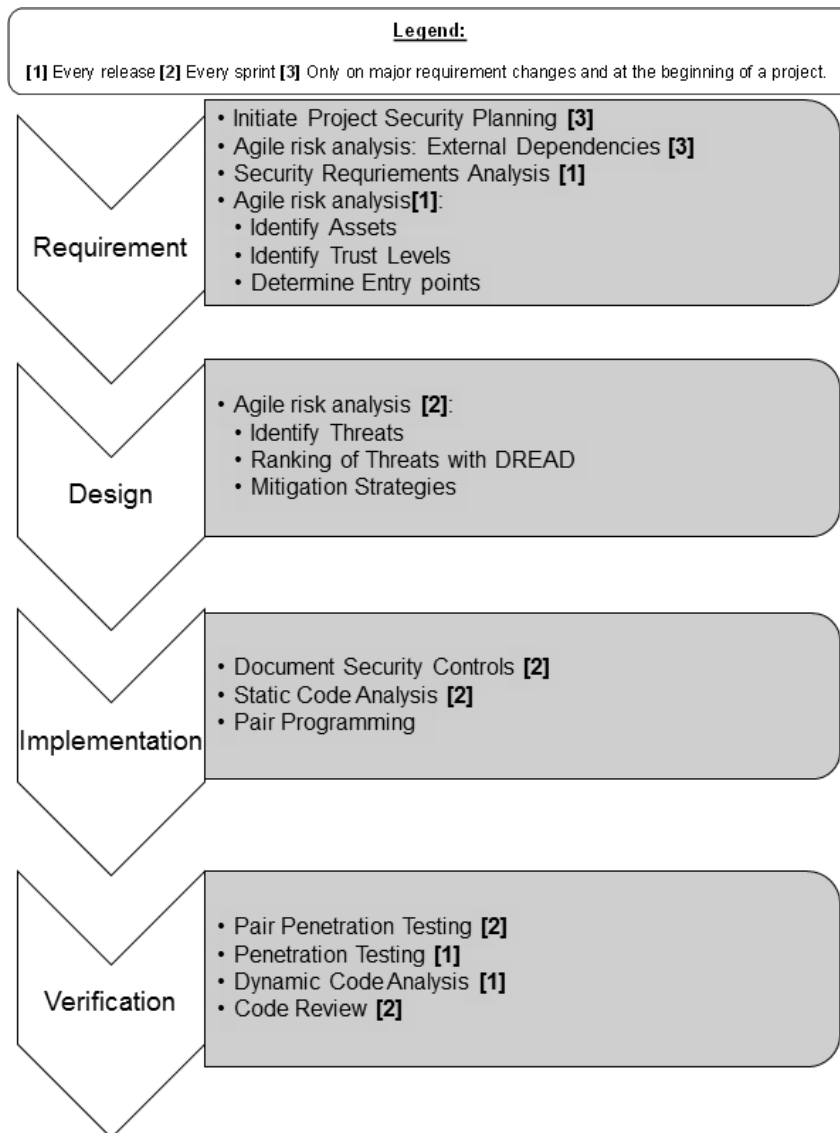


Figure 7.1: Waterfall structure of Secure Scrum process.

7.1.5 Quality Gates

Quality gates describe milestones in a software development process [91]. The aim is to evaluate whether all obligatory criteria in a software development phase, for example, requirement or design, have been fulfilled to proceed to the next phase. Therefore, they are located between phases. It is not recommended to move to the next phase before the requirements of the quality gate are met. All quality gates listed in this Section already exist in Scrum and are modified by us to aid the implementation of "Secure Scrum".

7.1.5.1 Definition of Ready

We add the risk analysis steps of the "Design" phase to this check-list. In this way, these steps have to be performed before the user story is implemented.

7.1.5.2 Definition of Done

We integrate all activities of the "Verification" phase to the "Definition of Done". Hence, for each user story that contains security requirements and for each security-related user story, all activities of the "Verification" phase have to be performed in order that this story receives the "Definition of Done". We adopted this idea to incorporate security verification criteria from [82].

7.1.5.3 Product Backlog Refinement

According to Scrum, the goal of this meeting, is that user stories which are not ready for implementation are refined. We add several risk analysis steps in this meeting in order to refine the user stories in regard to security. We extend this meeting in such a way that each user story should be refined in this meeting till such time as that it obtains the "Definition of Ready", which can require several iterations. If requirements of an user story change, in this meeting should be made a decision whether it still gets the "Definition of Ready". If it loses it, the story has to be refined again and the risk analysis steps have to be revisited.

7.2 Secure Scrum

Figure 7.3 shows a high level overview of the Secure Scrum process, where we integrate each activity of the waterfall process. This process is based on Scrum and the practices that we described in Section 3.2.4. All elements except the ones which we highlighted in red rectangles are part of the Scrum framework and these further practices. We modified each Scrum practice (are the green boxes in the Figure) in order to aid the implementation of Secure Scrum. We also adopt the principle of Release Planning, which we already explained in Section 3.2. For this process we assume that a company ships about four releases in one year. Each release consists of six Sprints and each is performed in the time-span of two weeks. Of course, a company can also ship more or fewer than four releases in a year, and also a Sprint can last shorter or longer than two weeks, nevertheless to simplify this process we assume it. The Figure is based on aspects of a system dynamics (SD). Figure 7.2 explains all of its elements:


<i>Element</i>	Name	Description
Release preparations	General phase of the process	Possible phases are: "Release preparations", "Deployment" or "Specification of epic or user story"
Sprint	Sprints	"Sprint before release" represents the Sprint before a release and "Sprint" represents all other sprints in a release
Sprint review	Practices of Scrum	Common Scrum practices such as: Sprint planning, Backlog refinement, Sprint review
"Definition of Done"	Quality gate	These elements represent quality gates of this process. They are connected to an action which means that they are located in this action or practice
Penetration testing	Security activities	Security activities are performed when an action is performed. For example: security activities of the implementation phase are performed when user stories are developed
"Definition of Ready" fulfilled	Outcome of an action or meeting	The outcome of an action or Scrum practice is described
	Action	Actions are performed to proceed to the next phase or Scrum practice

Figure 7.2: Components of the Secure Scrum Figure 7.3.

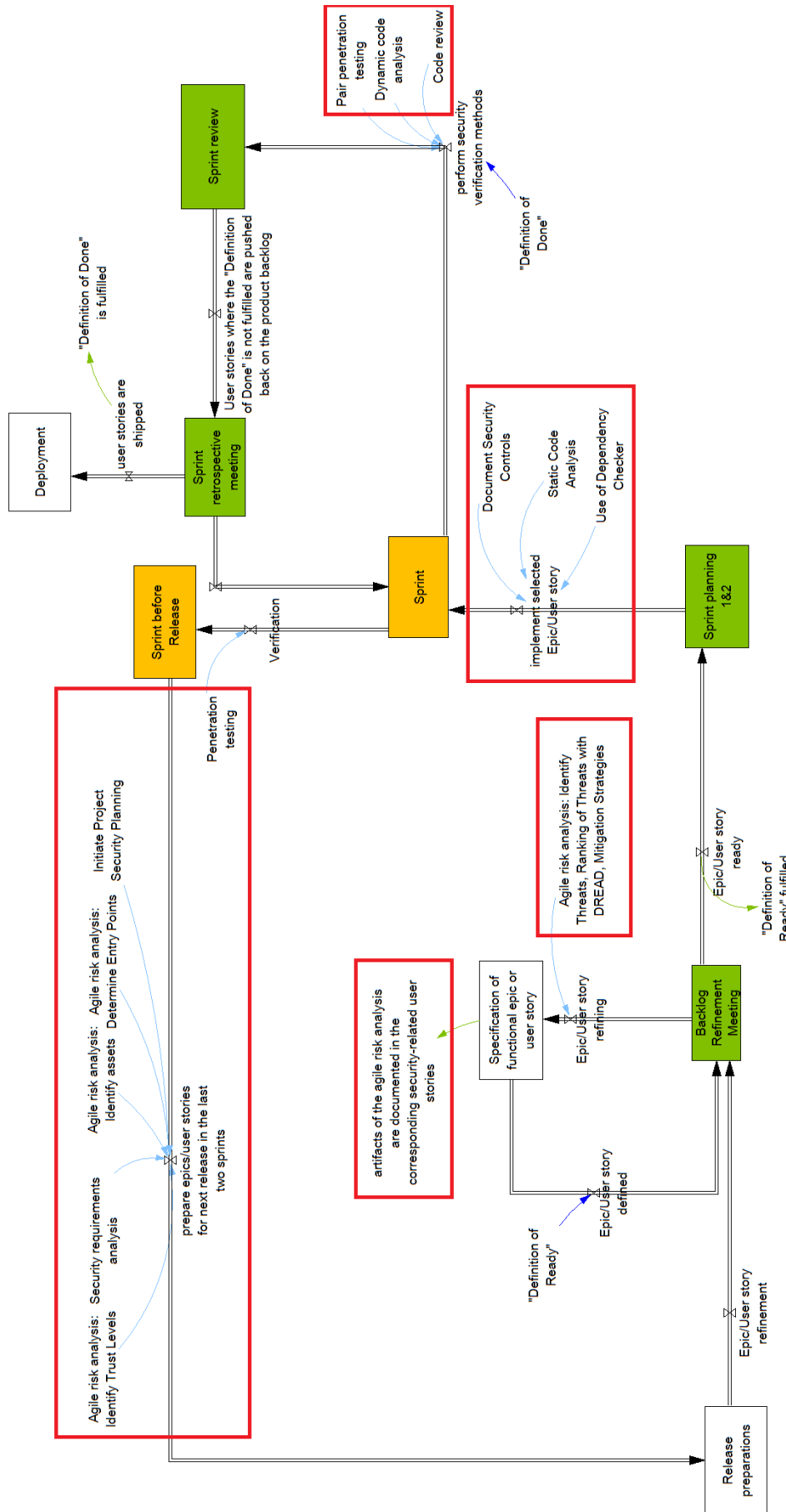


Figure 7.3: Secure Scrum process.

In the next few subSections we describe on a high-level our Secure Scrum process and its phases. First of all, we claim that a Security Risk Specialist performs a rough security requirement analysis with the help of the customer and Product Owners in the release planning phase. Next, the following phases are performed:

7.2.1 Requirement

In the current release (release n) the requirement phase of the next release (release n+1) is started, which means that in the last two Sprints of a release the epics and user stories of the next release are planned and also the identification of high-level security requirements, trust levels, possible assets and their entry points is performed. The artefacts of these points are included into a security Section of the epics or user stories of release n+1.

7.2.2 Design

The starting point of our next phase: "Design" is to define for each technical user story security-related ones. In normal Scrum the main reason why Product Backlog Refinement meetings are held is to evaluate whether user stories or epics have been changed and which have to be refined [88]. If requirements have changed, a decision whether it has to be refined again is performed. Furthermore, if new user stories or epics should be added to the Product Backlog is also decided in this meeting. In this process that is also the case and we claim that security requirements are also considered in these meetings. We further add a specification phase to Scrum in which all security activities of the "Design" phase, which can be seen in Figure 7.1, are performed. In this phase as many threats as possible, which can arise when a functional user story is implemented are identified. These threats are then formulated into security-related user stories. Our quality gate "Definition of Ready", which we explained in the Section above, makes our added steps mandatory. Thus, only if all activities of the requirement and design phase for a user story have been completed, this user story is allowed to be added to the Sprint backlog and the implementation of it can be started.

7.2.3 Implementation

Like in normal Scrum after user stories have been added to a Scrum team's Sprint backlog, the stories are implemented. We additionally add security activities of the "Implementation" phase, which can be seen in Figure 7.1. Moreover, we claim that for each security control, which was developed in the context of a user story a documentation is written and added in the corresponding security-related user story. If no security controls have been developed, the reasons are documented in the corresponding security-related user story.

7.2.4 Verification

In the next step, we claim that stories are tested with pair penetration testing, code reviews, whitebox security testing, and penetration testing. We describe them in greater detail in Section 7.3.4. After these activities have been successfully performed, our "Definition of Done" is fulfilled by the user story or epic. Next, our slightly modified Scrum Retrospective Meeting (see Section 7.1.3) is held and a Sprint Review. We did not add any steps to the traditional Sprint Review, therefore it is the same as in ordinary

Scrum. After these meetings, user stories which obtained the "Definition of Done" are shipped to the customer just like in normal Scrum.

7.3 Activities, tools and quality gates of this proposal

This Section lists and elaborates all activities of this process. We grouped the activities into the following waterfall-phases:

- Requirement,
- Design,
- Implementation, and
- Verification.

For each of these points we provide a subsection, where we explain all activities of them. We illustrate each activity in table format. The heading of the table represents the name of the activity. The first row represents in which phase of the process this activity is performed. The second row states the roles which are responsible for performing it whereas the third row displays a description of the activity. The last row states the expected output of this activity.

7.3.1 Requirement

This phase consists of the following activities, which are outlined in further detail in the next few subsections. Every subsection describes one of the following activities:

1. Initiate Security Planning,
2. Agile risk analysis: External Dependencies,
3. Security Requirement Analysis,
4. Agile risk analysis: Identify Trust Levels,
5. Agile risk analysis: Identify Assets, and
6. Agile risk analysis: Entry Points.

7.3.1.1 Initiate Project Security Planning

We adopt this activity from NIST 800-64. Nothing new is introduced to it by us. Therefore, we only summarize what should be performed in this activity.

Initiate Security Planning	
Phase of SDL:	Requirement
Responsible Roles:	Stake holders of the project, management personnel of the company, Product Owners, one or several Security Risk Specialists, and one or several software architects
Description:	The aim of this activity is that the company discusses with stake holders of the project about which security policies should be established, and which high-level security requirements should be taken into account. Furthermore, they discuss about the security vision of the project and define which roles are needed for the project. If major changes occur this activity should be revisited.

Expected output:	<ul style="list-style-type: none"> • All participants have obtained a basic knowledge of the security expectations. • Supporting documents, such as protocols of the topics which have been discussed in meetings. • A provisional schedule of security mile stones.
------------------	---

7.3.1.2 Agile risk analysis: External Dependencies

Agile risk analysis: External Dependencies	
Phase of SDL:	Requirement
Responsible Roles:	Security Risk Specialist, Developer
Description:	In this activity the "External Dependencies" step of our agile risk analysis methodology is performed as stated in Section 6.3. The goal is to define in which environment the software will be deployed and on which security assumption further steps proceed. Moreover, a table is created in which every assumption is listed
Expected output:	<ul style="list-style-type: none"> • All security assumptions are listed in a table. • If a security-related user story depends on a security assumption, the id of the assumption is documented in the story and vice versa.

7.3.1.3 Security Requirement Analysis

Security Requirement Analysis	
Phase of SDL:	Requirement
Responsible Roles:	Product Owner, Developer, Security Risk Specialist
Description:	In this activity security requirements are analysed and written down in the security Section of user stories or epics. In order to achieve this, we adopt the approach of [50] to derive them from functional requirements, laws, standards and security policies. Functional- and non-functional requirements are identified. Furthermore, also possible attack scenarios can be added to the user story. They are then used in further steps. Not all security requirements and attack scenarios can be determined in this phase. Hence, risk analysis steps are performed in order to elicit more specific security requirements.
Expected output:	<ul style="list-style-type: none"> • All participants obtained a high level understanding of security requirements of the epic or user story and functional- and nonfunctional security requirements are written down as acceptance criteria in the functional user story.

7.3.1.4 Agile risk analysis: Identify Trust Levels

Agile risk analysis: Identify Trust Levels	
Phase of SDL:	Requirement
Responsible Roles:	Quality Engineer, Product Owner, Developer
Description:	In this phase all possible users who have access rights to the application are analysed and also attacker profiles are created. This step is described in greater detail in Section 6.5
Expected output:	<ul style="list-style-type: none"> • High level understanding of who has access to the application has been obtained. • Database entries have for all users been created.

7.3.1.5 Agile risk analysis: Identify Assets

Agile risk analysis: Identify Assets	
Phase of SDL:	Requirement
Responsible Roles:	Security Risk Specialist, Developer
Description:	All possible assets are listed and categorized by trust levels, such as described in Section 6.4
Expected output:	<ul style="list-style-type: none"> • High level understanding of possible assets has been obtained. • All assets which are implemented in a user story are entered into the database and categorized on their impact levels.

7.3.1.6 Agile risk analysis: Entry Points

Agile risk analysis: Entry Points	
Phase of SDL:	Requirement
Responsible Roles:	Security Risk Specialist, Developer
Description:	Each entry point which is implemented in the user story is added into the database as described in Section 6.6. If anomalies are found in the database, such as that it does not make sense that a user has access to a specific asset, a threat is created. This threat is then handled in the “Agile Risk analysis: Mitigation Strategies” activity.
Expected output:	<ul style="list-style-type: none"> • High level understanding of possible entry points has been obtained. • All entry points have been inserted into our risk analysis database.

7.3.2 Design

The goal of this phase is the completion of the agile risk analysis on a low level view. After completing all activities described in this phase, all high level security details are specified in the user story and the

implementation can be started. It consists of the following activities, which we describe more elaborately in the next few Sections. Every Section describes one of the following activities:

1. Agile risk analysis: Identify Threats,
2. Agile risk analysis: Ranking of Threats with DREAD, and
3. Agile risk analysis: Mitigation strategies.

7.3.2.1 Agile risk analysis: Identify Threats

Agile risk analysis: Identify Threats	
Phase of SDL:	Design
Responsible Roles:	Development Team, Security Risk Specialist
Description:	In the refinement of a user story, the developers identify as much threats as possible in the context of a specific user story. In order to achieve this they perform the in Section 6.7 described steps. High level threats which have already been defined in previous steps are analysed whether they are specific enough for a story. If for any reason that should not be the case, they are defined as specific as possible. The number of involved developers can vary. We recommend that as much developer as possible should be part of this task. In this way, all developers are aware of possible threats which can occur when this user story is implemented. Furthermore, many points of views are included in the analysis. Security expertise can vary from person to person. It might be that one knows threats in one category which the other is not aware of. Discussions of several analysts about possible threats can therefore lead to better results. The next step is that the analysts define for each identified threat a security-related user story and attach abuse cases regarding this story into them.
Expected output:	<ul style="list-style-type: none"> • Threats which can arise if a user story is implemented are listed. • For each threat security-related user stories are created and abuse cases are attached to it. These user stories are then attached to their corresponding functional one. • Many developers of the team are now aware of which threats can arise if the user story is implemented.

7.3.2.2 Agile risk analysis: Ranking of Threats with DREAD

Agile risk analysis: Ranking of Threats with DREAD	
Phase of SDL:	Design
Responsible Roles:	Development Team, Product Owner

Description:	In this activity, the results of the analysis of the previous activity are presented to the whole team in another Product Backlog Refinement meeting. In these meetings, we claim that the Product Owner discusses with the whole team about the impact of each threat. For this reason DREAD is used as a supporting technology as described in Section 6.8. If for any reason the ranking cannot be completed for a user story, it obtains no “Definition of Ready”. Hence, the user story has to be refined again in another Product Backlog Refinement meeting.
Expected output:	<ul style="list-style-type: none"> • Every threat is ranked according to DREAD. • This rating is included in each security-related user story. • Many developers of the team are now aware of which threats can arise if the user story is implemented.

7.3.2.3 Agile risk analysis: Mitigation Strategies

Agile risk analysis: Mitigation Strategies	
Phase of SDL:	Design
Responsible Roles:	Development Team
Description:	After all threats have been ranked, for each of them the analysts perform the “Mitigation strategies” step, as described in Section 6.9 of our agile risk analysis methodology, in another Product Backlog Refinement meeting. The mitigation strategies are then documented in the corresponding security-related user stories.
Expected output:	<ul style="list-style-type: none"> • A mitigation strategy for every threat has been determined. • Every mitigation strategy has been formally written down in the corresponding security-related user story.

7.3.3 Implementation

This phase consists of the following activities, which are described in greater detail in the next few Sections. Every Section describes one of the following activities (every activity is performed at least once per Sprint):

- Document Security Controls,
- Static Code Analysis,
- Dynamic Code Analysis,
- Pair Programming, and
- Use of Dependency Checker.

7.3.3.1 Document Security Controls

Document Security Controls	
Phase of SDL:	Implementation
Responsible Roles:	Development Team
Description:	In our case the developers select one of the in “Agile risk analysis: Mitigation Strategies” identified security controls. These security controls are described only on a high-level. The developers should select them and tailor them in such a way that it fits for a specific user story. In the next step, they document in the corresponding security-related user story why they chose this specific security control. This documentation should be a short formal description.
Expected output:	The developers have identified a proper security control for a specific security-related user story and documented why it is sufficient.

7.3.3.2 Static Code Analysis

Static Code Analysis	
Phase of SDL:	Implementation
Responsible Roles:	Development Team
Description:	Every developer performs frequently security vulnerability scans with static code analysis tools on their written code. In order to do so the developers use approved tools. If issues in already existing code have been found, a security bug is immediately reported to a bug tracking system, such as Jira.
Expected output:	Code is approved by static security code analysis tools to be secure.

7.3.3.3 Dynamic Code analysis

Dynamic Code Analysis	
Phase of SDL:	Implementation
Responsible Roles:	Development Team
Description:	Approved dynamic code analysis tools are frequently used for testing the software.
Expected output:	Code is approved by well-established dynamic code analysis tools to be secure.

7.3.3.4 Pair Programming

Pair Programming	
Phase of SDL:	Implementation
Responsible Roles:	Development Team
Description:	We recommend performing pair programming for high-security critical software components. One developer should write the code, whereby the other one looks for coding failures or even security flaws in the code. In this way, security flaws can be mitigated early. Nevertheless, this requires that at minimum one has to have security expertise, since without it one does not recognize security flaws.
Expected output:	High-security critical code has been developed via pair programming.

7.3.3.5 Use of Dependency Checker

Use of Dependency Checker	
Phase of SDL:	Implementation
Responsible Roles:	Developer or Quality Engineer
Description:	In every Sprint a dependency checker tool is executed in order to check if libraries contain publicly-known security vulnerabilities. In case of unsecure libraries, a security bug is reported to a bug tracking system. If libraries with known issues are used, it is documented why it is not possible to use another library or whether the known issues are relevant. They can be irrelevant, due to, for example, the insecure functions of this library are not used in the project.
Expected output:	All possible security issues of used libraries are listed and documented on a central place like a TWIKI or Confluence page.

7.3.4 Verification

This is the last phase of the process. The verification process includes several testing and review steps. In the next few Sections we present the following activities:

- Pair Penetration Testing,
- Penetration Testing, and
- Code Review.

We claim that the verification process should be risk-based. Thus, the results of our agile risk analysis should be used to prioritize the test cases and reviews with regard to their risks. In this way, the security-critical parts of the software are tested first. This has the advantage that if time-issues occur, the high-critical parts have been tested first. We adopted this idea from Cigital Touchpoint. Additional to these activities, as we already discussed in Section 4.1.3.2, white-box testing, such as unit- and integration testing should be used as much as possible in order to assure that the software has no side-effects and security flaws. We recommend using the XP practice test-driven development to test the functionality of the whole software in an agile way. However, to test the functionality concerning security we claim that the Security Tester in the team writes these kind of tests, due to the fact that he has the most experience in security and knows common attack patterns, which he can test. We also recommend that these tests are automated as much as possible in order to be time-effective.

7.3.4.1 Pair Penetration Testing

This activity is a new one and introduced by us.

Pair Penetration Testing	
Phase of SDL:	Verification
Responsible Roles:	Quality Engineer, Developer

Description:	A Quality Engineer and a developer who developed a certain user story test with black box web vulnerability scanner (see Section 3.7.2 for greater details), whether the code that the developer wrote contains security flaws. If possible, only the new implemented functionality should be tested with this tool. For example, if a new site was implemented, only this site should be tested with the tool. In the next step the results of this analysis are triaged whether it contains false positives. If it contains false positives, both write a documentation, which lists all false positives that occurred and explain why they are no security flaws. The result is used to speed up testing in penetration testing, which is performed by a Security Tester.
Expected output:	<ul style="list-style-type: none"> • A Quality Engineer and the developer who implemented a user story conduct a vulnerability scan with the help of web-vulnerability scanner in order to approve that no security flaws have been inflicted. • Furthermore, they triage the results whether the code contains false positives and document them.

7.3.4.2 Penetration testing

Penetration Testing	
Phase of SDL:	Verification
Responsible Roles:	Security Tester
Description:	A penetration tester test either manually or automatically with all-in-one penetration testing tools attack scenarios. In order to first identify vulnerabilities that he can exploit to successfully perform an attack scenario, he can also use web-vulnerability scanner, such as in the previous activity "Pair Penetration Testing". However, this is not a mandatory step. The pen tester should use the list of false positives from the previous activity in order to be aware of which vulnerabilities he must not try to exploit. Issues detected are prioritized with DREAD and handled in the "Agile risk analysis: Mitigation Strategies" step.
Expected output:	Code which was implemented in the context of a user story is approved from a pen tester to be secure.

7.3.4.3 Code review

Code Review	
Phase of SDL:	Verification
Responsible Roles:	Developer, Security Tester or Security Risk Specialist
Description:	A Security Tester performs with the developer who developed a user story a secure code review. In this review they manually review the source code to identify whether the source code contains security flaws.
Expected output:	Source code is approved by a Security Tester to be secure.

7.4 How this process handles the PAs of the SSE-CMM

In this Section we elaborately discuss how this process takes the PAs of the SSE-CMM into account. This process proposal has a good coverage of every PA. Table 7.18 illustrates how our Secure Scrum process handles the different PAs. In the first row the PAs are listed, whereas in the second column we briefly explain how Secure Scrum conducts them. We do not consider PA 01 and 08, due to the fact they do not focus on the implementation and verification of secure software. To coordinate security we use the events of Scrum, as can be seen in Figure 7.3, and the frequent face-to-face interactions. We use our agile risk analysis to achieve the goals of PA 02–05. In this methodology we identify threats and vulnerabilities of the implemented software, assess the impact on assets and calculate the actual security risk with DREAD. Static- and dynamic code analysis as well as code reviews detect further vulnerabilities. In order to assure that the software meets the requirements of the customer (part of PA 06 and PA 10) in regard of security we involve the customer in defining the security milestones in the "Initiate Security Planning" activity and in the "Security Requirement Analysis" activity. In this way, it is assured that the requirements reflect the customer's wishes. In order to identify proper security control alternatives to satisfy the security requirements defined (PA 09) we conduct our agile risk analysis and use SAFECODE's security-related user story templates, which list several tasks that have to be considered if a functionality is implemented. In the "Document Security Controls" activity we select specific security controls and implement them. Finally, in order to assure that the selected security controls actually fulfil the security requirements defined (PA 06 and PA 11) we perform Pair Penetration Testing, Penetration Testing, Static Code Analysis, Code Review, and whitebox security testing. Additionally, we add one Security Risk Specialist and a Security Tester to each Scrum team and involve them in several security engineering activities in order to spread security knowledge among all developers.

Process Area (PA)	How Secure Scrum handles these PAs
PA 01: Administer Security Controls	Not considered.
PA 02: Assess Impact	Agile risk analysis method.
PA 03: Assess Security Risk	Agile risk analysis method.
PA 04: Assess Threat	Agile risk analysis method.
PA 05: Assess Vulnerability	<ul style="list-style-type: none"> • Agile risk analysis method, • "Static Code Analysis" activity, • "Dynamic Code Analysis" activity, and • "Code Review" activity,
PA 06: Build Assurance Argument	<ul style="list-style-type: none"> • The Customer and a security expert are involved in defining security requirements, acceptance criteria and security-related user stories. • "Security Requirement Analysis" activity, • "Initiate Security Planning" activity, and • Assurance methods of PA 11.
PA 07: Coordinate Security	The coordination of security is performed with Scrum events, and with face-to-face interactions.

PA 08: Monitor Security Posture	Not considered.
PA 09: Provide Security Input	<ul style="list-style-type: none"> • Developers are trained in security. • One security expert is added to each Scrum team. • Security-related user story templates are used to define for each functional user story security-related user stories. • "Document Security Controls" activity.
PA 10: Specify Security Needs	<ul style="list-style-type: none"> • "Initiate Security Planning" activity. • "Security Requirement Analysis" activity.
PA 11: Verify and Validate Security	<ul style="list-style-type: none"> • "Pair Penetration Testing" activity, • "Penetration Testing" activity, • "Static Code Analysis" activity, • "Dynamic Code Analysis" activity, • "Code Review" activity, • "Pair Programming" activity, and • Whitebox security tests, such as unit-, and system testing

Table 7.18: How Secure Scrum handles the Process Areas of the SSE-CMM.

Chapter 8

Evaluation

In this Chapter we perform two evaluations of our secure Scrum process. In the first we perform an evaluation with regard to the agile- and cost-effectiveness of secure Scrum, and how much additional time is needed to complete this process in contrast to ordinary Scrum. The aim of the second evaluation is to analyse which level of security it provides.

8.1 Evaluation of agility, cost-effectiveness, and additional time-effort

We conducted a survey in which we asked employees of a company with the help of a questionnaire how agile- and cost-effective each activity of our process they think it is, and how much time they would invest for completing each activity in the context of a user story. This company develops Java EE web applications for telecommunication companies and follows the Scrum process.

Interviewees

We explained the interviewees in a one hour presentation the whole secure Scrum process and described each of its activities. After the presentation, we answered participants' questions on subjects they were not familiar with and then ten employees filled out our questionnaire. Four developers, two Scrum Masters, one Software Architect, one Project Manager, one Quality manager, and one Product Owner participated in this survey. We involved people with different software engineering roles to cover as many viewpoints to a software development lifecycle as possible. We only selected employee who held the belief that they are able to estimate the time, cost, and agility of our secure Scrum process.

Brief description of the asked questions

First, we asked them which experiences they have already made regarding security in IT. Five employees stated that they have low security expertise, three claimed that they have medium knowledge of security and two said that they have high skills in security. We also asked them how much additional time effort would be acceptable for security in practice. From their point of view, the additional time-effort should not be more than 20%. Hence, our goal of this process is that it needs only about +20% time-effort.

We assume the following conditions in this questionnaire:

- A web-application is developed,
- It should be developed in such a way that it is secure concerning the OWASP Top 10,
- All interfaces of the web-application are publicly available and they are deployed on a public server,

- Only user stories which produce code are considered. Hence, user stories which focus, for example, on deploying the software or on the infrastructure are not taken into account,
- The time needed for planning, implementing, and testing a user story is about two weeks in ordinary Scrum. The employees in this company work 38,5 hours per week. Thus, 77 hours are needed to complete a user story. This criterion is especially important so that the participants can make rough time estimations for each activity of our secure Scrum process.

We asked how much time they would invest to perform each activity of the secure Scrum process. We also questioned how agile friendly, in their opinion, each activity is from a scale from one to five, whereby one is not agile-friendly. Furthermore, we asked how cost-effective each activity is. For these questions we provided the following selection criteria: very low, low, medium, high, and very high. Next, we questioned which activities are not needed, from their point of view. The main focus of the second section is to evaluate whether security related user story templates are a suitable concept for defining security requirements in an agile project. Finally, we were also interested in whether security-related user stories are accepted by them as good agile construct. Thus, we asked them whether security-related user stories should be backlog items like normal user stories with the answer possibilities yes or no. On the other hand, we also questioned whether instead of using security-related user stories security requirements should be added as acceptance criteria of normal user stories and a security section should be included in each functional user story, which describes abuse cases of this story. The next sections have the purpose to illustrate statistics of the results of our survey.

8.1.1 Results of first subsection of our questionnaire

8.1.1.1 Results of questions regarding security activities and tools

Figure 8.1 shows the estimate of time, cost, and agility that people in different software engineering roles gave. Figure 8.2 shows how the different security expertises answered our questions, whereas in Figure 8.3 we show the median scores of all participants' answers. The first two Figures either list the different software engineering roles or interviewees grouped by their security expertise. Each security expertise group and software engineering role contains three columns, such as time, cost, and agility. In the last figure we show the time, cost and agility column, as they present the overall results. The time column displays how much time in hours the participants would spend in average to complete an activity. The cost and agility columns illustrate the participants' median scores in regard to cost and agility. The maximum value in these columns is five, which represents that all participants have agreed that the activity is very agile. However, in contrast, if the value is one, all participants hold the belief that this activity is not agile-friendly. We use the following scale:

(1–1.49=very low, 1.5–2.49=low, 2.5–3.49=medium, 3.5–4.49=high, 4.5–5 = very high)

According to our results among all participants secure Scrum is on the scale of very low, low, medium, high, and very high, regarded as medium agile and cost-effective. In accordance with our results among all participants to complete each activity of secure Scrum about 34,7 (+45%) additional hours are needed in the context of a user story which needs in ordinary Scrum about 77 hours to plan, implement, and verify.

How the different software engineering roles evaluated the activities of secure Scrum Figure 8.4 states how the different software engineering roles evaluated all activities of secure Scrum in average in regard of agility, cost, and time whereas Figure 8.5 displays how the different security expertise groups rated them. The Software Architect holds the opinion that the process is “high” agile and “medium” cost-effective. The software developers, Product Owners, Scrum Masters, Quality Managers, and Project

Managers believe that the process is “medium” agile. By contrast, the software developers, Software Architects, Product Owners, Scrum Masters, and Quality Managers reckon that our secure Scrum process is “medium” cost-effective. The Project Manager holds the belief that our secure Scrum process is “low” cost effective. The employees with low, medium, and high security expertise agree that the process is “medium” agile and cost-effective.

Additionally, we asked the employees which activities should be removed from the process and further questioned the reason behind their answers. Several share the view that "Agile risk analysis: Identify trust levels" and "Dynamic Code Analysis" add no significant value compared to other activities. Furthermore, they believe that removing "Agile risk analysis: Identify Threats", "Agile risk analysis: Ranking of threats with DREAD" and "Document Security Controls" would simplify the process. On the other hand, some are of the opinion that pair penetration testing is a good approach and would thus remove penetration testing. They think that pair penetration testing is sufficient and is better than doing it alone.

Security activities evaluated by Role	Software Developer			Software Architect			Product Owner			Scrum Master			Quality Manager			Project manager			
	Time (hours)	Cost	Agility	Time (hours)	Cost	Agility	Time (hours)	Cost	Agility	Time (hours)	Cost	Agility	Time (hours)	Cost	Agility	Time (hours)	Cost	Agility	
Security Requirement Analysis	2,75	3	2,5	1	3	4	2	2	2	10	4	3,5	2	5	3	2	2	3	2
Agile risk analysis: Identify trust levels	1,5	2,5	2	1	2	3	2	2	2	2,5	4	3,5	2	4	2	2	2	2	2
Agile risk analysis: Identify assets	1,25	2	2,5	2	3	3	2	2	2	1,5	3,5	3,5	2	4	2	2	2	2	2
Agile risk analysis: Determine entry points	2,75	2,5	2,5	4	4	4	2	2	2	1,5	3,5	4	1	4	2	2	2	3	3
Agile risk analysis: External Dependencies	3	3	2,5	1	3	4	2	3	3	3	3,5	3,5	1	3	2	1	2	2	1
Agile risk analysis: Identify threats	2	2	3,5	2	2	2	2	2	2	2,5	2	3	1	3	2	1	2	2	4
Agile risk analysis: Ranking threats with DREAD	1	3	4,5	1	3	2	2	2	2	3	2	3	1	4	2	1	2	2	4
Agile risk analysis: Mitigation Strategies	4	3	3	4	4	4	2	3	3	2	3,5	3	2	3	2	2	2	2	3
Document security controls	3,5	2	2	2	3	3	4	1	1	1,5	4	2	2	3	2	2	2	2	2
Static- and dynamic code analysis	5,25	3	3	4	4	5	1	3	3	2,5	4	3	2	3	4	2	3	2	3
Pair programming		3,5	3,5		4	5		4	4		3,5	2,5		4	4			3	4
Pair penetration testing	6,5	3,5	4,5	4	3	4	2	4	4	4	3,5	4,5	2	4	4	2	2	2	4
Penetration testing	5,25	3	3,5	4	4	4	1	3	3	2,5	4	4,5	2	2	3	2	2	3	2
Code Review	3,5	3	3,5	2	4	5	2	3	3	2,5	3,5	3	2	4	4	1	1	3	4
Average	42,25	2,79	3,00	28,00	3,29	4,00	26,00	2,57	2,86	40,00	3,46	3,32	22,00	3,57	2,71	22,00	2,43	2,79	2,79

Figure 8.1: Evaluation results of each security activity of secure Scrum grouped by SE role.

Security activities evaluated by Security Expertise	Low			Medium			High		
	Time (hours)	Cost	Agility	Time (hours)	Cost	Agility	Time (hours)	Cost	Agility
Security Requirement Analysis	2,2	4,0	2,0	2,3	3,0	3,0	10,0	3,0	3,5
Agile risk analysis: Identify trust levels	1,8	3,0	2,0	2,0	2,0	3,0	1,5	3,0	3,0
Agile risk analysis: Identify assets	1,8	2,0	2,0	1,7	3,0	3,0	1,0	3,0	3,5
Agile risk analysis: Determine entry points	2,6	3,0	2,0	2,0	3,0	4,0	1,0	3,5	3,0
Agile risk analysis: External Dependencies	2,6	3,0	2,0	1,7	3,0	3,0	2,5	3,5	3,5
Agile risk analysis: Identify threats	1,4	2,0	3,0	2,3	2,0	3,0	2,5	2,0	3,0
Agile risk analysis: Ranking threats with DREAD	1,2	3,0	4,0	2,0	3,0	3,0	1,5	2,0	3,0
Agile risk analysis: Mitigation Strategies	3,6	3,0	3,0	2,0	3,0	3,0	3,0	3,5	2,5
Document security controls	2,6	2,0	2,0	4,0	3,0	2,0	1,0	3,5	2,0
Static- and dynamic code analysis	1,6	3,0	3,0	8,0	3,0	3,0	1,5	4,0	3,0
Pair programming		3,0	4,0		3,0	3,0		4,5	4,0
Pair penetration testing	2,4	4,0	4,0	8,0	3,0	5,0	4,0	4,0	4,5
Penetration testing	1,6	3,0	4,0	8,0	3,0	4,0	1,5	4,5	4,0
Code Review	2,2	3,0	2,0	3,3	3,0	4,0	2,5	3,0	2,5
Average	27,60	2,92	2,85	44,00	2,85	3,23	31,00	3,38	3,27

Figure 8.2: Evaluation results of each security activity of secure Scrum grouped by security expertise.

Security activities evaluated in average			
Security Activity or tool	Time (hours)	Cost	Agility
Security Requirement Analysis	3,8	3,5	3
Agile risk analysis: Identify trust levels	1,8	2,5	2
Agile risk analysis: Identify assets	1,6	2,5	2,5
Agile risk analysis: Determine entry points	2,1	3	3
Agile risk analysis: External Dependencies	2,3	3	2,5
Agile risk analysis: Identify threats	1,9	2	3
Agile risk analysis: Ranking threats with DREAD	1,5	3	3
Agile risk analysis: Mitigation Strategies	3	3	3
Document security controls	2,7	2,5	2
Static code analysis	3,5	3	3
Pair programming		3	3,5
Pair penetration testing	4,4	4	4,5
Penetration testing	3,5	3	4
Code Review	2,6	3	2,5
	Average	34,7	3

Figure 8.3: Evaluation results of each security activity of secure Scrum in average.

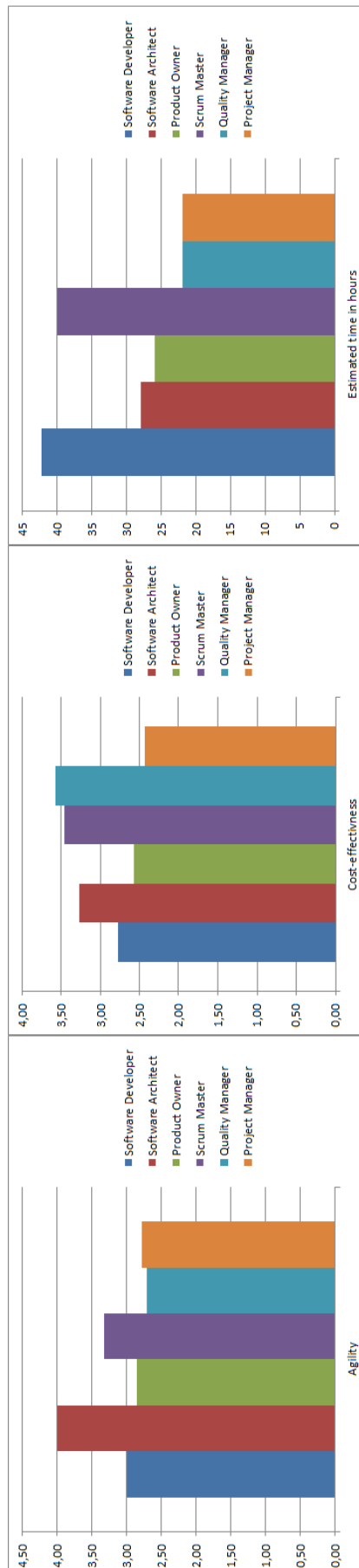


Figure 8.4: Evaluation results of all security activities of secure Scrum in average grouped by role.

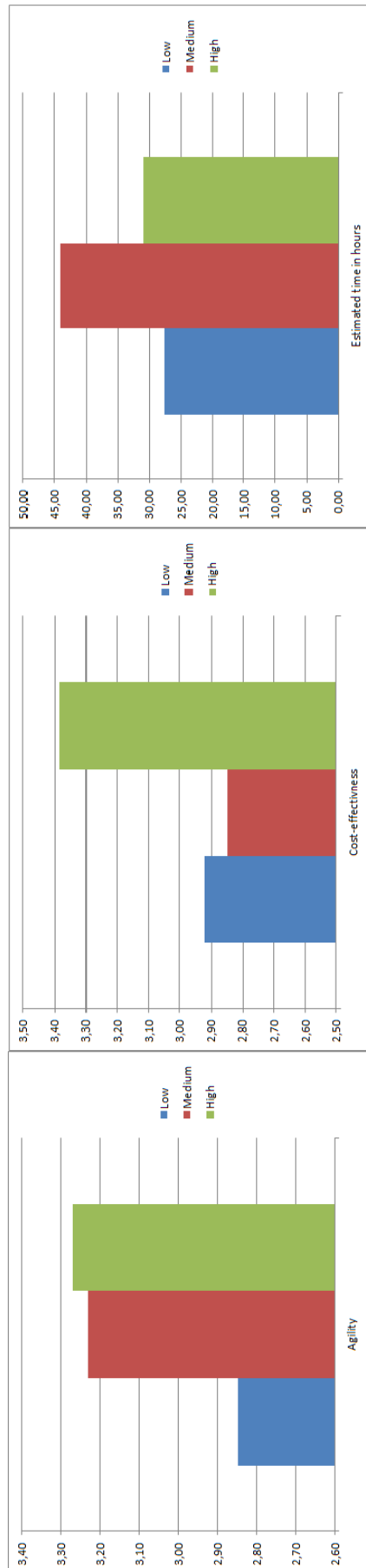


Figure 8.5: Evaluation results of all security activities of secure Scrum in average grouped by security expertise.

8.1.1.2 Results of questions regarding security-related user stories

Except one participant all agreed that security-related user stories should be added as Product- and Scrum backlog items. The one who does not agree stated that security-related user stories would allow for security measurements to be decoupled from their origin, reduced in priority and deferred until they are not realistic anymore.

8.1.2 Discussion of the results

We were surprised when evaluating the questionnaire, because everyone also ticked that in normal user stories security sections should be included where abuse cases are described and that security requirements should be added as acceptance criteria of normal user stories. We made a mistake in creating the question catalogue, since these two questions were supposed to be exclusive. We asked them why both ways should be used. They answered that on the one hand high security-critical requirements should be added as acceptance criteria and that abuse cases should be added in these stories. On the other hand, medium and small prioritized requirements should be formulated as security-related user stories and be attached to the normal user stories.

In the average opinion of the participants, the additional time needed for this new process is about 45% in contrast to ordinary Scrum. However, it is important to mention that those are only rough estimations and that +45% are only needed if this process is compared with an software development lifecycle based on Scrum where no time for security is invested at all, which is hopefully not the case in practice.

In summary, based on the evaluation of a group of developers, Scrum Masters, Software Architects, Project Managers, Quality Managers, and Product Owners on the scale of “very low”, “low”, “medium”, “high”, and “very high”, our secure Scrum process is regarded as “medium” agile and “medium” cost-effective. However, this is just the opinion of people who have seen our secure Scrum process proposal on paper. To receive more reliable evaluation results our secure Scrum process has to be tried out in practice.

8.2 Evaluation of Secure Scrum in regard of security with OWASP-SAMM

In this Section we perform an assessment of the SSE-CMM with the help of the OWASP-SAMM evaluation framework in order to evaluate the provided security of this process.

8.2.1 Brief description of OWASP-SAMM

OWASP SAMM [21] is a framework that can be used to evaluate an organization’s security process by calculating a maturity level which represents the provided security of software security practices. The outcome of this assessment is a maturity level (a grading from 0–3) for several categories with regard to security. An interview where several yes-no questions in the categories: governance, construction, verification, and operations are asked serve as the basis of this assessment. For each question OWASP gave some guidance in what steps the process should be perform in order that the answer can be yes.

The governance category focuses on questions with respect to whether developers are trained in security, whether all parties are aware of the compliance requirements, and whether assurance programs are in place. (The governance category consists of the following subcategories: Strategy & Metrics, Policy & Compliance, and Education & Guidance)

The construction category aims on questions regarding whether threat assessments are performed, if security requirements are specified, and whether a secure architecture is built. (The construction category

consists of the following subcategories: Threat Assessment, Security Requirements, and Secure Architecture)

The verification category focuses on questions with respect to whether design- and implementations reviews are performed, and how the security is tested. (The verification category consists of the following subcategories: Design Review, Implementation Review, and Security Testing)

The operations category aims on questions regarding whether issues are handled, if an incident response plan is in place, and whether the applications is securely deployed and maintained. (The operations category consists of the following subcategories: Issue Management, Environment Hardening, and Operational Enablement)

We do not evaluate the process in the categories governance and operation, due to our process only focus on the development and the verification of secure software. The categories' construction and verification have both three sub categories. Each sub category can receive 0-3 points, where zero states that the process does not support this category and three that it provides full support. There are also half points, which is presented with a "+", for example, "2+" means 2,5.

8.2.2 Evaluation with OWASP-SAMM

We conducted this assessment and filled out the questionnaire. Figure 8.7 shows our security assessment of secure Scrum with OWASP SAMM in the Threat Assessment subcategory, Figure 8.8 in the Security Requirements subcategory, Figure 8.9 in the Security Architecture subcategory, Figure 8.10 in the Design Review subcategory, Figure 8.11 in the Implementation Review subcategory, and Figure 8.12 in the Security Testing category. The first column contains OWASP SAMM's assessment questions of each subcategory. For each question OWASP gives some guidance in what steps should be performed in the process in order that the answer can be yes. The second column shows our answers to the question. In the third column we describe why we answered a question with yes.

8.2.3 Results and discussion of results

As can be seen in Figure 8.6 our process reaches full support in almost all subcategories, except in Secure Architecture and Design Review. Therefore, this process is capable of developing secure software. Our process lacks support in two out of six sub-areas. Both categories focus on a secure software design. The aim of the secure architecture subcategory is to give the team guidance in how a secure design of an information system should look like with, for example, design patterns. On the other hand, the focus on the design analysis is to perform secure design reviews in order to validate whether the architecture of the software is secure.

We intentionally did not include any activity which focuses on this area, because in our process, writing each user story developers have to think which threats can occur when they implement a certain functionality and how these threats can be mitigated. Thus, they actually have to think about how they design it to be secure and therefore we think that a separate activity where explicitly a secure design is created and reviewed is not necessary. We think that the process would then have too much planning overhead.

Construction	Threat Assessment	3			
Construction	Security Requirements	2+			
Construction	Secure Architecture	1			
Verification	Design Analysis	0			
Verification	Implementation Review	3			
Verification	Security Testing	3			

Figure 8.6: Security assessment of secure Scrum with OWASP SAMM.

Construction			
	Threat Assessment	Yes/No	Interview Notes
TA1	Do projects in your organization consider and document likely threats? <i>Guidance:</i> Likely worst-case scenarios are documented for each project based on its business risk profile. Attack trees or a threat model is created for each project tracing the preconditions necessary for a worst-case scenario to be realized. <i>Guidance:</i> Attack trees or threat models are expanded to include potential security failures in current and historical functional requirements. When new features are added to a project, attack trees or threat models are updated.	Yes	Threats to the application are determined in our agile risk analysis method.
	Does your organization understand and document the types of attackers it faces? <i>Guidance:</i> Potential external threat agents and their motivations are documented for each project. Potential internal threat agents, their associated roles, and damage potential are documented for each project or architecture type. <i>Guidance:</i> A common set of threat agents, motivations, and other information is collected at the organization level and re-used within projects.	Yes	
TA2	Do project teams regularly analyze functional requirements for likely abuses? <i>Guidance:</i> Each project derives abuse-cases from its use-cases. As project requirements or features are added, abuse-cases are updated.	Yes	In our agile risk analysis abuse cases are determined for each functional user story. Threats are prioritized with DREAD. The stake holders of the project are involved in defining the security requirements. We involve the customer and stake holders in Secure Scrum's "Initiate Project Planning" and "Security Requirement Analysis" activity.
	Do project teams use a method of rating threats for relative comparison? <i>Guidance:</i> A documented weighting system based on documented threat agents, exploit value, technical difficulty, and other factors is used to rank threats. Remediation of vulnerabilities is prioritized based on the weighting system.	Yes	
	Are stakeholders aware of relevant threats and ratings? <i>Guidance:</i> Potential threats and ratings are reviewed with project stakeholders.	Yes	
TA3	Do project teams specifically consider risk from external software? <i>Guidance:</i> Third-party, external libraries and code used in each project are clearly identified and documented for each project. Project threat models are updated based on identified threat agents and motivations for third party libraries and code.	Yes	In Secure Scrum's "Dependency Checker" activity will be evaluated whether the software implemented contains vulnerable libraries. Mitigation strategies are identified in Secure Scrum's "Agile risk analysis: Mitigation Strategies" step.
	Are the majority of the protection mechanisms and controls captured <i>Guidance:</i> An assessment for each project has been conducted to identify mitigating controls that prevent preconditions identified in attack trees or threat models. This assessment is updated each time new features or requirements are introduced or the attack tree is modified. Mitigating controls have been documented within the attack tree or threat model. Mitigating controls or security requirements have been added to each project to address any preconditions that still lead to a successful attack within attack trees.	Yes	

Figure 8.7: Security assessment of secure Scrum with OWASP SAMM (threat assessment category).

Security Requirements		Yes/No	Interview Notes
SR1	Do project teams specify security requirements during development? <i>Guidance:</i> Security requirements are derived from functional requirements and customer/organization concerns. <i>Guidance:</i> A security auditor leads specification of security requirements within each project. <i>Guidance:</i> Security requirements are specific, measurable, and reasonable. <i>Guidance:</i> Security requirements are documented for each project.	Yes	Security requirements are identified in Secure Scrum's "Security Requirement Analysis" activity.
	Do project teams pull requirements from best practices and compliance guidance? <i>Guidance:</i> Industry best practices are used to derive additional security requirements. <i>Guidance:</i> Existing code bases are analyzed by a security auditor for opportunities to add security requirements. <i>Guidance:</i> Plans to refactor existing code to implement security requirements are prioritized by project stakeholders including risk management, senior developers, and architects.	Yes	
SR2	Do stakeholders review access control matrices for relevant projects? <i>Guidance:</i> Users, roles, and privileges are identified in each project. <i>Guidance:</i> Resources and capabilities are identified in each project. <i>Guidance:</i> A matrix of roles and capabilities is documented for each project. <i>Guidance:</i> As new features are introduced, the matrix documentation is updated. <i>Guidance:</i> The matrix is reviewed with project stakeholders prior to release.	Yes	For each functionality which is implemented in a user story that allows user to access a particular asset through a specific entry point an entry in our agile risk analysis database is added.
	Do project teams specify requirements based on feedback from other security activities? <i>Guidance:</i> Additional security requirements are created based on feedback from code reviews, penetration tests, risk assessments, or other security activities.	Yes	
SR3	Do stakeholders review vendor agreements for security requirements? <i>Guidance:</i> During the creation of third-party agreements, specific security requirements, activities, and processes are considered for inclusion.	No	Penetration Testing is continuously performed.
	Are audits performed against the security requirements specified by project teams? <i>Guidance:</i> Audits are routinely performed to ensure security requirements have been specified for all functional requirements. <i>Guidance:</i> Audits also verify attack trees are constructed and mitigating controls are annotated. <i>Guidance:</i> A list of unfulfilled security requirements and their projected implementation date is documented. <i>Guidance:</i> Security requirement audits is performed on every development iteration prior to the implementation of code.	Yes	

Figure 8.8: Security assessment of secure Scrum with OWASP SAMM (security requirements category).

Secure Architecture		Yes/No	Interview Notes
SA1	<p>Are project teams provided with a list of recommended third-party components?</p> <p><i>Guidance:</i> A weighted list of commonly used third-party libraries and code is collected and documented across the organization. The libraries are informally evaluated for security based on past incidents, responses to identified issues, complexity, and appropriateness to the organization. Risk associated with these components are documented.</p> <p><i>Guidance:</i> A list of approved third-party libraries for use within development projects is published.</p>	Yes	In Secure Scrum's "Dependency Checker" activity will be evaluated whether the software implemented contains vulnerable libraries.
	<p>Are project teams aware of secure design principles and do they apply them consistently?</p> <p><i>Guidance:</i> A list of secure design principles (such as defense in depth) have been collected and documented. These principles are used as a checklist during the design phase of each project.</p>	Yes	
SA2	<p>Do you advertise shared security services with guidance for project teams?</p> <p><i>Guidance:</i> A list of reusable resources is collected and categorized based on the security mechanisms they fulfill (LDAP server, single sign-on server, etc.). The organization has selected a set of reusable resources to standardize on. These resources have been thoroughly audited for security issues. Design guidance has been created for secure integration of each component within a project. Project groups receive training regarding the proper use and integration of these components.</p>	No	
	<p>Are project teams provided with prescriptive design patterns based on their application architecture?</p> <p><i>Guidance:</i> Each project is categorized based on architecture (client-server, web application, thick client, etc.). (Risk-based authentication system, single sign-on, centralized logging, etc.). Architects, senior developers, or other project stakeholders identify applicable and appropriate patterns for each project during the design phase.</p>	No	
SA3	<p>Do project teams build software from centrally-controlled platforms and frameworks?</p> <p><i>Guidance:</i> Reusable code components based on established design patterns and shared security services have been created for used within projects across the organization. Reusable code components are regularly maintained, updated, and assessed for risk.</p>	No	
	<p>Are project teams audited for the use of secure architecture components?</p> <p><i>Guidance:</i> Audits include evaluation of usage of recommended frameworks, design patterns, shared security services, and reference platforms. Results are used to determine if additional frameworks, resources, or guidance need to be specified as well as the quality of guidance provided to project teams.</p>	No	

Figure 8.9: Security assessment of secure Scrum with OWASP SAMM (security architecture category).

		Verification	
Design Review		Yes/No	Interview Notes
DR1	<p>Do project teams document the attack perimeter of software designs?</p> <p><i>Guidance:</i> Each project group creates a simplified one-page architecture diagram representing high-level modules. Each component in the diagram is analyzed in terms of accessibility of the interface from authorized users, anonymous users, operators, application-specific roles, etc.</p> <p><i>Guidance:</i> Interfaces and components with similar accessibility profiles are grouped and documented as the software attack surface.</p> <p><i>Guidance:</i> One-page architecture diagram is annotated with security-related functionality.</p> <p><i>Guidance:</i> Grouped interface designs are evaluated to determine whether security-related functionality is applied consistently.</p> <p><i>Guidance:</i> Architecture diagrams and attack surface analysis is updated when an application's design is altered.</p>	No	
	<p>Do project teams check software designs against known security risks?</p> <p><i>Guidance:</i> Each project group documents a list of assumptions the software relies on for safe execution.</p> <p><i>Guidance:</i> Each project group documents a list of security requirements for the application.</p> <p><i>Guidance:</i> Each project's one-page architecture diagram is evaluated for security requirements and assumptions. Missing items are documented as findings.</p> <p><i>Guidance:</i> Evaluations are repeated when security requirements are added or the high-level system design changes occur within a project.</p>	No	
DR2	<p>Do project teams specifically analyze design elements for security mechanisms?</p> <p><i>Guidance:</i> Each interface within the high-level architecture diagram is formally inspected for security mechanisms (includes internal and external application tiers). Analysis includes the following minimum categories: authentication, authorization, input validation, output encoding, error handling, logging, cryptography, and session management.</p> <p><i>Guidance:</i> Each software release is required to undergo a design review.</p>	No	
	<p>Are project stakeholders aware of how to obtain a formal secure design review?</p> <p><i>Guidance:</i> A process for requesting a formal design review is created and advertised to project stakeholders.</p> <p><i>Guidance:</i> The design review process is centralized and requests are prioritized based on the organization's business risk profile.</p> <p><i>Guidance:</i> Design reviews include verification of software's attack surface, security requirements, and security mechanisms within module interfaces.</p>	No	
DR3	<p>Does the secure design review process incorporate detailed data-level analysis?</p> <p><i>Guidance:</i> Project teams identify details on system behavior around high-risk functionality (such as CRUD of sensitive data). Project teams document relevant software modules, data sources, actors, and messages that flow between data sources or business functions.</p> <p><i>Guidance:</i> Utilizing the data flow diagram, project teams identify software modules that handle data or functionality with differing sensitivity levels.</p>	No	
	<p>Does a minimum security baseline exist for secure design review results?</p> <p><i>Guidance:</i> A consistent design review program has been established. A criteria is created to determine whether a project passes the design review process (for example no high-risk findings). Release gates are used within the development process to ensure projects cannot advance to the next step until the project successfully completes a design review.</p> <p><i>Guidance:</i> A process is established for handling design review results in legacy projects, including a requirement to establish a time frame for successfully completing the design review process.</p>	No	

Figure 8.10: Security assessment of secure Scrum with OWASP SAMM (design review category).

Implementation Review		Yes/No	Interview Notes
IR1	<p>Do project teams have review checklists based on common security related problems?</p> <p><i>Guidance:</i> The organization has derived a light-weight code review Developers receive training regarding their role and the goals of the checklist.</p>	Yes	<p>In Secure Scrum's "Code Review" activity a security specialist reviews the code whether the code meets the security requirements defined.</p>
	<p>Do project teams review selected high-risk code?</p> <p><i>Guidance:</i> High-risk software components are prioritized above other code during the review process. Security auditors or code review tools are utilized to perform the checklist based code review. <i>Guidance:</i> Remediation of findings in high-risk components are prioritized appropriately.</p>	Yes	
IR2	<p>Can project teams access automated code analysis tools to find security problems?</p> <p><i>Guidance:</i> The organization has reviewed open source, commercial, and other solution for performing automated code reviews and selected a solution that will best fit the organization. <i>Guidance:</i> Automated code analysis has been integrated within the development process (at code check-in for example).</p>	Yes	<p>Static- and dynamic code analysis tools are used in Secure Scrum.</p> <p>Security flaws detected in code reviews are handled as threats. In Secure Scrum's "Agile risk analysis: Mitigation Strategies" mitigations strategies for this threat are decided.</p>
	<p>Do stakeholders consistently review results from code reviews?</p> <p><i>Guidance:</i> Project stakeholders review and accept any risks that they Project stakeholders have created a plan for addressing findings <i>Guidance:</i> in legacy code.</p>	Yes	
IR3	<p>Do project teams utilize automation to check code against application-specific coding standards?</p> <p><i>Guidance:</i> Automated code review tools are customized to perform additional API checks, verify organization coding standards, etc. (including the addition of custom rules).</p>	Yes	<p>Static- and dynamic code analysis tools are used in Secure Scrum.</p> <p>Secure Scrum extends the "Definition of Done" in such a way that a specific level of code review results have to be met.</p>
	<p>Does a minimum security baseline exist for code review results?</p> <p><i>Guidance:</i> Each project contains a checkpoint in the development process that requires a specific level of code review results to be met before release. The organization has established an exception process for legacy code, which requires a certain level of assurance to be met within a specific time period <i>Guidance:</i></p>	Yes	

Figure 8.11: Security assessment of secure Scrum with OWASP SAMM (implementation review category).

	Security Testing	Yes/No	Interview Notes
ST1	Do projects specify security testing based on defined security requirements? <i>Guidance:</i> The organization has documented general test cases based on security requirements and common vulnerabilities. Each project has documented test cases for security requirements specific to that project. <i>Guidance:</i> Staff ensures test cases are applicable, feasible, and can be executed by relevant development, security, and quality assurance staff.	Yes	In the Sprint Retrospective meeting security-related user stories which are likely to occur in further development are archived. Furthermore, the test-cases used to ensure that the security requirements defined in these security-related user stories are met are also archived.
	Is penetration testing performed on high risk projects prior to release? <i>Guidance:</i> Project teams engage security auditors to perform penetration testing. <i>Guidance:</i> Penetration testing covers security requirements and test cases at a minimum. <i>Guidance:</i> Penetration testing issues are resolved to an acceptable level of risk prior to release.	Yes	In Secure Scrum penetration testing is performed.
	Are stakeholders aware of the security test status prior to release? <i>Guidance:</i> Penetration testing issues are reviewed with project stakeholders. <i>Guidance:</i> Project stakeholders select issues to remediate prior to release. Project stakeholders set a time line for addressing identified issues or accept outstanding risks.	Yes	Issues detected in penetration testing are prioritized with DREAD and handled in the "Agile risk analysis: Mitigation Strategies" step.
ST2	Do projects use automation to evaluate security test cases? <i>Guidance:</i> The organization has reviewed open source, commercial, and other solution for performing automated security testing and selected a solution that will best fit the organization. <i>Guidance:</i> Automated security testing has been integrated within the	Yes	In Secure Scrum, automated testing, such as unit-, and integration testing is performed. Furthermore, web vulnerability scanning is performed and static- and dynamic code analysis tools are used.
	Do projects follow a consistent process to evaluate and report on security tests to stakeholders? <i>Guidance:</i> Automated security testing occurs across projects on a regular, scheduled basis. <i>Guidance:</i> A process has been created for reviewing security testing results with project stakeholders and remediating risk.	Yes	Security tests are performed regularly.
ST3	Are security test cases comprehensively generated for application-specific logic? <i>Guidance:</i> Using automated tools, unit tests, or other similar methods, a comprehensive set of security test cases is constructed and evaluated for each project.	Yes	In Secure Scrum, automated testing, such as unit-, and integration testing is performed. Furthermore, web vulnerability scanning is performed and static- and dynamic code analysis tools are used.
	Does a minimum security baseline exist for security testing? <i>Guidance:</i> Each project contains a checkpoint in the development process that requires a specific level of security testing results to be met before release. <i>Guidance:</i> The organization has established an exception process for handling security testing results in legacy projects, which requires a certain level of assurance to be met within a specific time period	Yes	Secure Scrum extends the "Definition of Done" in such a way that a specific level of security testing has to be performed.

Figure 8.12: Security assessment of secure Scrum with OWASP SAMM (security testing category).

Chapter 9

Summary and Conclusion

The aim of this master thesis was to show a possible way of how to derive a Scrum-compatible secure software development process and to propose a secure Scrum process that has a good coverage of all PAs of the SSE-CMM while maintaining the flexibility and efficiency provided by an agile software development process. We achieved both goals by analysing security activities of well-established non-agile SE processes in regard of agility and by integrating the most agile-friendly security activities into Scrum.

9.1 Summary of methodology

In this thesis we first reviewed the SSE-CMM to identify which tools, methodologies, techniques, and practices can be integrated into Scrum in order to ensure that all process areas of the SSE-CMM are fulfilled. Second, we analysed which ones can fit into an agile development process and we evaluated for this reason how agile security activities of NIST 800-64, SREP, Cigital Touchpoint, and Microsoft SDL are with the method described in [46]. In this method Keramati and Mirian-Hosseiniabadi [46] stated that nine *agile features*, including features such as speed of execution, people orientation, and change tolerance should be used to evaluate how agile security activities are. However, Keramati and Mirian-Hosseiniabadi [46] never stated what the exact meaning of a *agile features*. Therefore, we specified the meaning of these *agile features* and defined a rough metric which helped us to evaluate how agile friendly these activities are. As a next step, we assessed the agility of each activity and discussed why we chose a specific grade in a particular *agile feature*.

After we analysed these activities, we recognized that all of the threat modeling-, and risk analysis activities assessed are not agile-friendly. These activities cover exactly the PA 02 ("Assess Impact"), PA 03 ("Assess Security Risk"), PA 04 ("Assess Threat"), and PA 05 ("Assess Vulnerability") of the SSE-CMM, which no other activity analysed does. Therefore, we proposed a more agile risk analysis method by combining ideas of several agile friendly activities and by using OWASP's Application Threat Modeling as reference. Next, we integrated all agile friendly activities and our agile risk analysis into Scrum, created a secure Scrum process, and illustrated each of its phases. Finally, we conducted an evaluation of how agile our secure Scrum process is and how much security it provides. In order to evaluate the secure Scrum process's agility, cost-effectiveness and roughly estimated additional time effort we conducted a survey in which we asked ten employees of a company which develops JAVA EE web applications with Scrum how agile friendly, cost-effective and time consuming the newly introduced practices, activities and tools are. In order to evaluate the security provided for our process we assessed it with OWASP-SAMM. We will discuss each of the contributions in the following subsections.

9.1.1 Analysis of already existing security activities

By analyzing the security activities of SREP, Microsoft SDL, Cigital Touchpoint, and NIST 800-64, we reached the conclusion that threat modeling, and risk analyses are not agile-friendly. Furthermore, manual testing techniques, such as risk based- and red team testing are not agile-friendly. In summary, about half of the activities analysed do not fit into agile. For many other activities (about 25%), such as Abuse Cases, Attack Surface Reduction, and Penetration Testing we actually found no real conclusion whether they are agile friendly, because they did not reach our desired threshold of about 67,5%, but had more than 50% of all *agile features*, which can be interpreted in the way that most of the agile values are fulfilled. We decided to categorize them as activities that can fit into agile if they are slightly tailored. In further steps of this thesis we modified several of them to fit into Scrum or we only took several parts of it or only the main idea of them. For example, we integrated Abuse Cases, the idea of an Attack Surface Analysis, Critical Assets, and Categorize Information System into our agile risk analysis method. As further example, we added manual Penetration Testing into our secure Scrum process without modifying it. However, we decided that it should not be performed for every user story, but rather frequently throughout the software development lifecycle. On the other hand, we actually found activities which fit well into agile. The highlights are, Security Requirement Analysis, Static and Dynamic Analysis, and Code Reviews. In summary, only about 25% analysed security activities actually fit into agile. Without the other ones we were not able to create a secure Scrum process which takes all PAs of the SSE-CMM into account.

9.1.2 Novel proposal for agile risk analysis (ARA)

Our agile risk analysis aims to aid the nature of agile that software is implemented in iterations. We created this methodology that it should be performed for each user story, rather than performing it just once at the beginning of a project, because in each user story assets, trust levels, and entry points can be added, modified, or removed. In our methodology, before a user story is implemented the developers have to identify assets, trust levels, and entry points which this user story adds, modifies, or removes and they have to insert them into a database described by us. We presented how this database scheme looks like using the following example. It shows which trust levels have which access restrictions on each asset and from which entry point they can access it. Furthermore, it illustrates which user story implemented this functionality. This information is needed in order that the analysts know which user stories they have to revisit for changes on assets, trust levels, or entry points. We claim that this analysis helps developers recognize which assets they have to protect. Second, it supports that they realize from which entry points assets are accessible and to get them familiar with whether several entry points are really necessary to add. For example, during assessing the impact on assets they realize which assets they have to lay an eye on. Further, while identifying entry points of assets they see whether it would be better to unify several entry points or to remove some to reduce the attack surface. We decided to use a database scheme, because we wanted that this information can be quickly queried in further analyses in order that everyone stays on top of changes on assets, entry points and trust levels which occur when user stories are developed.

As next step, threats are analysed in our methodology, from our point of view, in a light weight way, because no DFDs have to be drawn in contrast to Microsoft's Threat Modeling or OWASPs solution. We decided to adopt the solution of using security-related user story templates of SAFECode, since we think that with these many threats can be analysed with minimal time effort. These templates describe what developers, testers, and quality engineers should do to defend against the OWASP Top 10. They are written in a defender's viewpoint, because they describe what should be done to develop a secure application in order that common security issues (threats can here be seen as weaknesses of security controls) do not occur in the first place. However, in our opinion, also the attacker's viewpoint of threats should be considered in order that the analysts know against which concrete attack scenarios they have the application to defend against. Thus, we claim that the analysts get more familiar with how they can

design specific test scenarios and it is easier for them to rate these threats, because they actually know how a hacker can attack the system and what the impact is. Therefore, we suggest describing abuse cases for each security-related user story template, to map attacks of the OWASP Top 10 document to them, and to include them into the corresponding security-related user story. In the sixth step of our methodology we claim that threats are rated with DREAD of Microsoft in order that the analysts know the most critical ones and are able to decide which threats or weaknesses they have to consider. Of course, they can also decide to not act, for example, because the impact of a threat is minimal or the likelihood of concurrence is low. If they decide to actually mitigate an identified threat, the specified fundamental practices of SAFECode in their security-related user story templates are used, the from SAFECode provided CWE links and also the security controls which are described in the OWASP Top 10 document.

9.1.3 Our secure Scrum process

We achieved our goal to propose a secure Scrum process that covers all PAs of the SSE-CMM while maintaining the flexibility and efficiency provided by an agile software development process.

We evaluated the agility of security activities of existing non-agile security engineering processes in order to know which activities can fit into an agile development process. We integrated only the most agile-friendly ones into Scrum. If we have not found any agile-friendly activity which takes a PA of the SSE-CMM into account, we chose activities which we were not sure of whether they are actually agile-friendly and modified them to better fit into agile. As discussed in the Section above, in order to take the PAs of the risk process of the SSE-CMM into account we created our own agile risk analysis method. In this way our secure Scrum proposal fulfils all PA goals of the SSE-CMM while maintaining the flexibility and efficiency provided by an agile software development process. As can be seen in Section 8.2, our process also supports many categories of the OWASP-SAMM evaluation framework for secure software life cycles. It only lacks support for a secure architecture and reviewing this secure architecture. We intentionally did not include any activity which focus on this area, because in our process, writing each user story developers have to think which threats can occur when they implement a certain functionality and how these threats can be mitigated. Thus, they actually have to think about how they design it to be secure and therefore we think that a separate activity where explicitly a secure design is created and reviewed is not necessary. We think that the process would then have too much planning overhead.

We think we showed a quite useful way of what a secure Scrum process can look like and we showed that security can be integrated into Scrum without losing too much agility and without extensively additional time-effort.

Chapter 10

Further work

Although we think that we created a useful and secure Scrum process there is still much work in this area.

10.1 Analysis of SE processes in regard of agility

In further work, security activities of other SE processes, such as OWASP CLASP [76], can also be evaluated in regard of agility. Maybe activities of other SE processes are more agile than the ones we analysed.

Our approach to assess the agility of security activities was to evaluate the grade of which they support agile features by using our introduced metric in Chapter 5. However, for several agile features, such as change tolerance and speed of execution, a better approach would have been if a Scrum team had tried these activities out in practice. For example, the agile feature change tolerance was hard for us to evaluate, due to the fact that we did not try these activities out in practice, but rather analysed it from a theoretical perspective. In this case, we evaluated how many subtasks of a security activity have to be redone on small or medium software requirement changes and whether it depends on other activities. We tried to specify in greater detail what small or medium software requirement changes actually are which means that at maximum 50% of the software requirements can change. Nevertheless, we think that these changes should be specified in much greater detail. We could have received more reliable results if a Scrum team had tried all security activities out when developing a web application by following the Scrum framework. Beforehand, user stories and security requirements would be formulated and discussed with the Scrum team, which they would have to implement. Next, concrete changes would have to be defined at the beginning, but should only be discussed with the team during different phases of the software development lifecycle to simulate software requirement changes in practice. Finally, the time to start over security activities can have been measured. Additionally, the speed of execution of each activity could be measured if a team tried the activities out.

10.2 Agile Risk Analysis (ARA)

The main reason why our secure Scrum process has a quite high planning overhead is, because we integrated our agile risk analysis into it. However, we think that we have achieved a quite agile way of addressing the identification of threats, vulnerabilities, risks, and impacts, because our methodology mainly consists of agile-friendly security activities. We have three points which can be done in further work.

- The first is to create a tool which provides a graphical user interface for our database scheme.

- The second is to evaluate this methodology and the tool in practice, to see whether the time needed to perform it.
- Furthermore, our method can be evaluated in how much threats can be found with it compared to other methods, such as Microsoft's Threat Modeling. To evaluate how many threats can be identified and mitigated with it two teams with approximately equal security expertise can be setup to develop the same web application with the same functional- and security requirements. The first develops the software with, for example, Microsoft's Threat Modeling and the second with ARA. For evaluating the provided security of the web applications, for example, a red team can test for both how many vulnerabilities they find and how much attacks would be successful.

10.3 Secure Scrum Process

The same principle as in the section above can also be used to evaluate our whole secure Scrum process. The first team can follow, for example, the Cigital Touchpoint SE process instead of just using threat modeling. The second should then apply our secure Scrum process. The evaluation would be exactly the same as explained in the section above for evaluating how long it takes to develop the application and how secure it is.

Appendix A

Evaluation of security activities of SE processes

In this Chapter we analyse each security activity which we explained in Section 5.2 with respect to how agile they are in greater detail. Therefore, in this Chapter we only analyse these security activities. A description of each activity can be found in Section 5.2. The order of the activities listed in this chapter is the same as in Section 5.2. We assume for this chapter that the reader is familiar with each security activity. Furthermore, we discuss each and why it got a specific grade concerning a particular *agile feature*. In order to make it more readable we use a table-format for the evaluation of each activity. The headline of the table represents the name of the activity. In the next seven rows we discuss each *agile feature* and why we give it a concrete grade. We assign for each *agile feature* a grade in the interval [0, 5]. The agility degree of an activity is the sum of all measured *agile feature* grades. If this value is high, the activity can be integrated into Scrum and - if not that means integrating it would be difficult, because it is not “agile-friendly”. In each row we added a rating column which displays which grade we gave this activity. The last row displays the agility degree of an activity in a red font.

A.1 Security activities of Microsoft SDL

Security Requirement Analysis		Rating
Simplicity:	In our opinion, the one who conducts this analysis must at minimum have an elementary overview over the basic security principles, such as confidentiality, integrity and availability. Moreover, one has to have domain specific knowledge in order to define use scenarios and has to know about some security policies, common threats and vulnerabilities.	2
Free of Modeling and Documentation:	In this activity nothing is implemented or tested.	0
Change Tolerance:	At most 50% of all security requirements are changed, according to our specification.	3
Speed of Execution:	-	4
Informality:	Without performing this activity the actual security needs of the customer are not known, such as security policies or security standards.	5
People Orientation:	Most of the work consists of brainstorming about security requirements in use scenarios.	4

Iterative / Dividable	Security requirements are derived from the answers of the provided security requirements questionnaire, from use scenarios, and from secure code reviews. Secure code reviews can be conducted iteratively and also use scenarios can be defined one by one. Another method for identifying more specific security requirements in the SDL is by performing threat modeling. Use scenarios are first formulated in the first step of the threat modeling process and then in further steps of this process vulnerabilities and threats are identified and prioritized. From the results of threat modeling more specific security requirements can be derived.	5
Flexibility:	Only the high important security requirements could be defined.	4
Agility Degree:		27

Role Matrix		Rating
Simplicity:	The work consists of identifying all user roles of the application. Furthermore, in this activity the users' access level to the application is determined. From our point of view, this analysis does not need a person with expertise in IT-security.	5
Free of Modeling and Documentation:	In this activity nothing is implemented or tested.	0
Change Tolerance:	According to our definition of change tolerance, at maximum, about 50% of the functionality the users have access to can change. This implies that only about 50% of the users and their access restrictions can be dropped or changed. Furthermore, this activity does not depend on other activities of the SDL.	3
Speed of Execution:	In our opinion, new user roles are not added in every user story. Therefore, in average only several users are added in the context of a user story.	4
Informality:	With this knowledge threats can be categorized more effectively, because one knows which users would be affected if a specific asset would be attacked.	4
People Orientation:	We think that user roles and their access level to the application should be discussed with the customer and derived from functional requirements. Thus, people interactions is needed for this activity.	4
Iterative / Dividable	User roles and functionality is added iteratively in agile development. Therefore, this activity should be conducted iteratively.	5
Flexibility:	User roles can be grouped to minimize the documentation effort. Furthermore, only the access restrictions to security critical assets could be determined.	4
Agility Degree:		29

Design Requirements		Rating
Simplicity:	We reckon that the person who performs this activity should have practical knowledge about how secure software should be designed.	1
Free of Modeling and Documentation:	Nothing is implemented or tested.	0

Change Tolerate:	If security- or functional requirements change, in our opinion, the design may change dramatically. For example, if in an e-shop web application a new requirement is added that credit card data has to be stored according to the PCI DSS standard the design has to be changed dramatically. Moreover, if the design has already been implemented the source code has to be modified too, which causes much additional work.	2
Speed of Execution:	Designing secure software may need a considerable amount of time to be completed, nevertheless, this could vary from user story to user story.	2
Informality:	The authors of [34] state that by conducting this activity many security flaws can be prevented. According to OWASP-SAMM, which is a reference framework for an secure software development lifecycle, a secure design is one of the most important points to consider for the development of a secure software.	4
People Orientation:	To create a secure software design secure design principles should be followed. To illustrate this secure design, several diagrams, such as UML and class diagrams have to be modelled. These steps are in our opinion rather method based. However, we think that also brain storming with other software developers about a secure software design is required to complete this activity.	3
Iterative / Dividable	From our point of view, it does not make sense to perform this activity for each user story, because in the context of a single user story only a small amount of functionality is implemented. We think that a secure software design should be rather modelled for large software components (user stories which are belonging together) in order that the designer has a bigger view of the functionality and thus can model security controls which fit for several other user stories as well. With this bigger view, the designer is also capable of designing layered defences. To summarize, we think that the work can be divided, but should not be performed for each user story. A secure software design should be modelled for several user stories which are belonging together.	4
Flexibility:	The creation of a secure software design could be conducted only for high-risk functionality.	4
Agility Degree:		20

Cost Analysis (By conducting a Product Risk Assessment)		Rating
Simplicity:	This activity does require a security and privacy team which perform security risk assessments and privacy impact ratings. We think that they should have practical security knowledge on what security controls are needed for the software which they want to develop. Furthermore, they should know the system's level of vulnerability in regard of security.	2
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	In this activity decisions are made whether and to which degree activities, such as threat modeling, security design review, penetration- and fuzz testing are conducted. Thus, the decisions made in this activity have impact on many other activities.	0
Speed of Execution:	The analysis is, in our opinion, rather light-weight, due to only questions are asked and analysed.	4
Informality:	This activity helps to roughly identify the effort, time and cost needed for the security related part of a user story. However, no threats, vulnerabilities or requirements are identified in the course of this activity.	3

People Orientation:	The work consists of asking questions regarding possible vulnerabilities of a system and which security components are needed. These questions are taken from a predefined question catalogue. The answers are then analysed by applying general defined rules and based on the expertise of the analysts. Based on these answers the effort for securely designing, and implementing the software is roughly estimated.	4
Iterative / Dividable	In our opinion, this activity can be performed for each user story to identify, for example, whether threat modeling is necessary for a user story.	4
Flexibility:	The aim of this activity is to determine on a high-level how much effort should be investigated in securing parts of the software. In agile software components are developed in increments (user stories). To achieve the goal to determine which security activities are required for which software components this cost analysis has to be performed for each user story. However, the analysts are rather flexible in deciding how much effort should be investigated to secure a portion of software.	2
Agility Degree:		14

Threat Modeling		Rating
Simplicity:	The one who conducts Threat Modeling has to model a DFD with the help of Microsoft's Threat Modeling tool. The tool automatically displays threats on a high-level based on the DFD with STRIDE-per-interaction, such as that the interaction between a process and a data store yields threats in the categories spoofing, and denial of service. In order to identify further threats and vulnerabilities, we think that one has to have practical security expertise and has to know about many common vulnerabilities, threats and attacks.	1
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	Changes in security- and functional requirements may require reviewing many threat models and reviewing decisions made from the threats identified. It may be necessary to update DFDs and to address further attack scenarios which could now occur, because of the changes in security- and functional requirements. For example, if a requirement is added to store credit card numbers securely a lot of new threats would arise. Furthermore, these changes may also require to implement further security controls, and to prioritize some threats again with DREAD.	2
Speed of Execution:	The decomposition of the software, drawing of diagrams, identifying of threats with STRIDE, finding of security controls, and rating them with DREAD requires in our opinion an considerable amount of time even in the context of a single user story.	2
Informality:	If it is performed properly one knows the high-risk threats and vulnerabilities of the software and knows how to mitigate them.	4

<p>People Orientation:</p>	<p>The using of Microsoft’s Threat Modeling Tool to draw DFDs and to generate high-level threats from these diagrams is from our point of view method-based. Also DREAD is rather method based, because specific questions are asked.</p> <p>However, to identify further threats, to determine attack patterns, to prioritize threats with DREAD requires, in our opinion, brain storming sessions with other developers.</p>	<p>2</p>
<p>Iterative / Dividable</p>	<p>Threat Modeling can be divided into the following steps: (1) Define use scenarios, (2) Gather a list of external dependencies, (3) Define security assumptions, (4) Create external security notes, (5) Create one or more DFDs of the application being modeled, (6) Determine threat types, (7) Identify the threats to the system, (8) Determine risk, (9) Plan mitigations.</p> <p>We reckon that each step is rather heavy-weight, therefore we give this activity only four points in this category.</p>	<p>4</p>
<p>Flexibility:</p>	<p>Threat Modeling could be used only for high-critical security components which were identified in the Product Risk Assessment of Microsoft SDL. However, this requires that a Product Risk Assessment was conducted, otherwise only the obvious high-critical security components are known.</p> <p>DFDs can be modelled on a high-level or in great detail with Microsoft’s Threat Modeling tool. In order that the tool can generate as specific threats as possible the DFD has to be drawn in as much detail as possible.</p>	<p>3</p>
<p>Agility Degree:</p>		<p>18</p>

<p>Attack Surface Analysis / Reduction</p>		<p>Rating</p>
<p>Simplicity:</p>	<p>We reckon that the person who performs this activity has to know common attacks, where the software can be attacked, has to know the principle of least privilege, and how layered defenses and access restrictions can be applied.</p>	<p>2</p>
<p>Free of Modeling and Documentation:</p>	<p>Documentation tasks of this activity:</p> <ul style="list-style-type: none"> • Design software for minimal attack surface with models, such as DFDs. • Review data flow diagrams and graphic presentations of the code in order to identify paths that does not require authentication. <p>This activity includes the following configuration/implementation points:</p> <ul style="list-style-type: none"> • Use the least privilege principle for privilege of processes and services, • Code or services that are not needed should be shut down or removed, • Entry points which are not needed should be removed, • Authentication and access restrictions should be implemented, • The access to network endpoints should be limited (authentication should be used). 	<p>3</p>

Change Tolerate:	<p>Software has already been implemented: In this case, if requirements change only the design has to be changed and eventually remodeled if some graphical representation has already been created. We assume that only small or medium changes occur, therefore only several parts of the design has to be remodeled.</p> <p>Software has not been implemented yet: In this case, the same as in the case above has to be done and additionally it may be necessary to add authentication or access control mechanism, reconfigure software components or to rewrite some code, which causes much additional effort.</p>	3
Speed of Execution:	-	3
Informality:	By performing this activity the attack surface is reduced and one knows where the vulnerable entry points to the software are located. The principle of least privilege is used and therefore entry points are limited and access restrictions are implemented.	3
People Orientation:	As can be seen in row two, the main work of this activity consists of methods and principles that should be applied. However, from our point of view, some brain storming sessions with other people are also needed to identify the attack surface.	2
Iterative / Dividable	<p>This activity can be divided in several steps which can be seen in the second row.</p> <p>In agile, software is implemented in increments (user stories). In each user story new user roles and entry points can be added and the attack surface can grow, hence it does make sense to conduct this activity for each user story.</p>	4
Flexibility:	We reckon that at minimum this activity should be performed for components or modules of the software that have open interfaces to be effective, due to a single vulnerable entry point is enough for an attacker to launch successfully perform a cyber-attack. However, in agile projects it does not have to be performed for every user story. Only for those which create, for example, new entry points or add new paths that are not protected by authentication.	3
Agility Degree:		23

Static Analysis		Rating
Simplicity:	The analysis is usually performed by normal developers. In order to filter false positives basic security knowledge is required, due to the fact that the analyst has to know whether the vulnerability identified from the tool is actually a real security flaw. Therefore, we reckon that the analyst has to have at minimum basic security knowledge.	3
Free of Modeling and Documentation:	The analysis is performed automatically. However, the results have to be reviewed manually and the findings have to be marked as false or true positives.	3

Change Tolerate:	Only if code changes, the tool has to be performed again. Many of these tools perform dataflow and controlflow analyses in order to minimize the false positive rate. In this case, the whole code base has to be analysed in order that it recognizes where, for example, malicious input has already been escaped [48]. If only several parts of the software are scanned with the tool it cannot detect whether malicious input has already been escaped in other classes and might therefore detect a false positive. This means that the entire code base has to be scanned in order that the tool is able to recognize all implemented safeguards. However, tools such as, Coverity SAST [95] allow to perform an incremental analysis of the code [28]. This means that the tool is able to analyse several parts of the software in increments. It is able to save results of previous runs and uses them to recognize the code in, for example, other modules. With this knowledge it can perform a dataflow- and controlflow analysis of the full code, which has already been analysed. For example, if the complete software is analysed in a nightly full-scan, then, when a developer makes changes in only several parts of the software the tool is able to use the results of the nightly full-build to recognize all implemented security controls in the whole code base. In this way, developers only have to scan files where they actually performed changes instead of analyzing the entire source code.	5
Speed of Execution:	According to [28], the analysis of several millions lines of code need only about as long as to build it, which is actually quick. In the case of tools which are capable of performing iterative analyses the execution time is drastically reduced, due to the fact that only several parts of the software have to be analysed.	5
Informality:	According to [73], these tools are able detect security flaws, such as XSS, XPATH Injection, SQL Injection, Path Traversal, and Command Injection. The authors of [73] evaluated several state-of-the-art static code analysis in how effectively these tools identify vulnerabilities in the source code. They state that tools which identify a high amount of true positives (>80%) unfortunately also detect about 50% false positives, which is quite high. Therefore, much additional effort is needed to filter these false positives manually.	2
People Orientation:	The analysis is performed automatically, but the results have to be reviewed manually.	3
Iterative / Dividable	Due to the fast speed of execution and the fact that this kind of tool can be run repeatedly in nightly builds, we give this activity five points in this <i>agile feature</i> .	5
Flexibility:	First, the source code has to be analysed with a source code analysis tool and then false positives have to be filtered manually. Furthermore, a full scan has to be performed frequently in order to resolve the dependencies of single classes that have changed for the dataflow- and control flow analysis. In our opinion, none of these steps can be skipped; otherwise the analysis cannot be completed. For example, if false positives are not filtered the real security flaws are not known.	1
Agility Degree:		27

For this evaluation we only consider the tool “Application Verifier” of Microsoft, due to the authors of [34] explicitly mentioned this dynamic analysis tool.

Dynamic Analysis	Rating
------------------	--------

Simplicity:	In our opinion, a normal developer is sufficient for this analysis, because only subtle programming errors, such as synchronization-, buffer overflow errors, and memory leaks can be detected with this kind of tool and no web vulnerabilities, such as SQL Injection, or XSS are discoverable [34, p. 265–268]. By contrast, static code analysis tools as mentioned in the “Static Analysis” activity are additionally able to detect web vulnerabilities, such as SQL Injection, and XSS. For these tools the analyst has to be aware of these attacks in order that he is capable of recognizing whether the vulnerability found by the tool is a false positive.	4
Free of Modeling and Documentation:	First, this tool has to be executed on a running system either manually or automatically on hand of some test-cases. It then displays at runtime, for example, synchronization problems when they occur. As next step, a human has to triage the results and eventually fix them immediately or he could also, for example, report the flaws as bugs. The reporting process would require some documentation.	4
Change Tolerate:	As mentioned above, these tools are capable of running with automatic test-suites. Thus, if requirements change and test-cases are rewritten nothing else has to be done on the part of this tool. If new tests are written it has to be configured to run automatically when they are executed. Nevertheless, if tests are modified or some new ones are added a tester has to analyse if new flaws occurred.	5
Speed of Execution:	Since these tools can be integrated into automatic tests the execution time is minimal, because of the fact that they can be run over night. However, an analyst has to analyse the results of it, which could require some time. For only several classes the effort should be reasonable. We estimate that about 1-2 hours are needed in the context of a user story to test the flaws, which the tool identified. In this time estimation we do not consider the time required to fix all determined bugs.	5
Informality:	This tool identifies only subtle programming flaws, such as buffer overflows, synchronization failures and memory corruptions. Hence, for example, web vulnerabilities, such as Cross-Site Request Forgery, or Path Traversal are not detectable with it.	2
People Orientation:	Coding errors can be found automatically with the help of this tool, but a human has to analyse the flaws in greater detail and has then to fix it.	2
Iterative / Dividable	In regard of the fact that it can be integrated into automatic test-suites the analysis can be divided into several parts.	5
Flexibility:	As mentioned in the “Change Tolerate” row this kind of tool can be configured to be executed automatically when automatic test-cases are performed. Therefore, one could argue that it provides much flexibility; due to it can be run on single test-cases. However, the disadvantage of this tool is that in order to be able to detect all flaws in the source code the program has to be run on every possible path, which needs a lot of testing time. This means that in order that it can detect as much defects as possible the tools have to be run on as much test-cases as possible. If, for example, only tests are written for the good behavior, these tools would not detect possible flaws in bad-paths, which an attacker might use to attack the system. Thus, in our opinion it provides not much flexibility, because many test cases have to be generated to test all possible paths.	1
Agility Degree:		29

Fuzz Testing		Rating
Simplicity:	These tools can be executed automatically, but one with some security knowledge has to identify the vulnerability in the code which causes the system crashes.	3
Free of Modeling and Documentation:	The crashed test cases have to be analysed and the vulnerabilities which have been found have to be documented, prioritized and reported.	3
Change Tolerate:	Only if code has changed the tool has to be run again.	5
Speed of Execution:	Can vary greatly. The run-time depends on the number of test cases that are executed and on the complexity of the system. However, this tool can be run automatically therefore it can be run over night or on weekends.	3
Informality:	By using such kind of tool one knows whether a web application is robust and is vulnerable to malformed input or known-to-be-dangerous values or files. Identifying the vulnerability in the code is out of the scope of black-box fuzzy testing tools.	3
People Orientation:	For example, the fuzzing tool JBroFuzz [70] of OWASP can be used automatically or semi-automated. In the semi-automated case, one has to select certain payloads in order to test against a particular set of vulnerabilities, such as SQL Injections.	2
Iterative / Dividable	Can be performed site per site, or for single requests, urls, forms, user-generated content, RPC requests, and more.	5
Flexibility:	This tool can be used, for example, for single requests, or urls therefore one is rather flexible in using this tool.	4
Agility Degree:		28

Code Review (Manual Approach)		Rating
Simplicity:	In order to analyse code whether it is secure one has to know what to look for. In our opinion, this person should have practical experience in security, has to know common attacks, vulnerabilities and code failures.	2
Free of Modeling and Documentation:	The code is tested (reviewed). Identified vulnerabilities have to be fixed immediately or be documented and eventually but not necessarily be prioritized.	4
Change Tolerate:	If security requirements change assumptions that the reviewer made when he reviewed the code could be incorrect. If these assumptions were documented only the list of assumptions has to be reviewed again. Otherwise, the analyst has to review the code again.	3
Speed of Execution:	From our point of view, in the context of a single user story the source code can be reviewed fast, because only several classes and methods have to be reviewed.	5
Informality:	One knows whether the source code is secure (depends on the security expertise of the reviewer).	4
People Orientation:	One or more person reviews the source code manually. A list of common failures and mitigations could be used to help the reviewer identifying vulnerabilities.	4
Iterative / Dividable	Method per method can be reviewed.	5

Flexibility:	Only high-risk components, classes, or methods could be reviewed.	5
Agility Degree:		32

A.2 Security activities of Cigital Touchpoint

Security Requirement Analysis		Rating
Simplicity:	Similar to Microsoft's Security Requirement Analysis.	2
Free of Modeling and Documentation:	Similar to Microsoft's Security Requirement Analysis.	0
Change Tolerate:	Similar to Microsoft's Security Requirement Analysis.	3
Speed of Execution:	-	4
Informality:	Similar to Microsoft's Security Requirement Analysis.	5
People Orientation:	The works consists only of brain storming about security requirements and deriving them from several sources.	5
Iterative / Dividable	For each user story or epic, security requirements can be analysed. If a user story has impact on other ones it may require updating security requirements of previous user stories. It could also be the case that a new user story depends on previous ones and can inherit some requirements from them. Thus, we think that security requirement analyses can be performed iteratively.	5
Flexibility:	Similar to Microsoft's Security Requirement Analysis.	4
Agility Degree:		28

Abuse Cases		Rating
Simplicity:	We reckon that the analyst has to know common attacks and vulnerabilities in order to identify abuse cases. One should also have practical experience in how the software could be attacked and how these attacks can be mitigated.	2
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	If requirements models have to be updated, removed or eventually new ones have to be added.	2
Speed of Execution:	Thinking about possible attacks and modeling and documenting them requires, in our opinion, some time even in the context of a single user story. For example, if a user story adds functionality that the credit limit of customers can be edited, many abuse cases can emerge.	3
Informality:	By conducting this activity one knows what an attacker could attack and how it could be mitigated. Furthermore, one has a graphical representation of these attacks.	4
People Orientation:	The work consists of thinking about how the system could be attacked and to model and document these cases.	2
Iterative / Dividable	Abuse cases can be added for each user story one by one (iteratively). However, prior to the implementation phase the developers should be aware of the abuse cases in order that they can plan mitigations to these attack scenarios.	5

Flexibility:	This activity could be performed only for high-risk components where user stories add functionality that has access to critical assets.	4
Agility Degree:		22

Architectural Risk Analysis		Rating
Simplicity:	The authors of [50, p. 147–148] stated that this activity is knowledge intensive and requires being familiar in using of attack patterns, and exploit graphs. Furthermore, the analyst should know about weaknesses in famous frameworks.	1
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	Similar to Threat Modeling of Microsoft.	2
Speed of Execution:	-	2
Informality:	Similar to Threat Modeling of Microsoft.	4
People Orientation:	This approach is, in our opinion, method based, because diagrams have to be drawn, exploit graphs should be modeled and abuse cases have to be mapped. However, several steps also contain thinking about vulnerabilities and threats by using checklists. Furthermore, especially the ambiguity analysis phase requires people interaction, because two or more people should analyse possible threats with their own methods and should then together discuss their findings.	3
Iterative / Dividable	This methodology can be divided into three steps: (1) attack resistance analysis, (2) ambiguity analysis, and (3) a weakness analysis.	4
Flexibility:	In contrast to Microsoft’s Threat Modeling we reckon that this approach offers more flexibility, because the authors of [50] mentioned several ways of how the above listed points could be performed. For example, security flaws could be detected with either checklists or historical knowledge or methods like STRIDE. A further example is that only high risk abuse cases could be mapped, which are known beforehand, due to the activity: “Abuse Cases”.	4
Agility Degree:		20

Manual or semi-automatic Penetration Testing		Rating
Simplicity:	A security expert is needed who can hack a system [49].	1
Free of Modeling and Documentation:	The main work of this activity consists of hacking an application and to identify vulnerabilities of it. However, the identified vulnerabilities have to be documented in some way and therefore this activity also requires some documentation work.	4
Change Tolerate:	If code changes we reckon that the software has to be tested again. For example, if a single method of a web application was changed which is used in many sites all these sites have to be tested again, because of the black box nature. This modified code could have side effects.	2
Speed of Execution:	Attacking a system and finding of vulnerabilities needs, in our opinion, some time.	2
Informality:	By conducting this activity one knows the weaknesses of the software. (Depends on the expertise of the pen tester)	4
People Orientation:	One or more people can conduct penetration testing of a single target application. The work consists of identifying vulnerabilities with tools or writing exploitation code and to think about attacks.	3

Iterative / Dividable	We reckon that several pen testers could attack the system on different entry points. Hence, the work can be divided.	4
Flexibility:	We reckon that penetration testing is rather flexible. For example, in case of testing a web application, the pen tester could only test several sites of a web application, which contains entry points to critical assets. Hence, the pen tester could only test the high-security critical components of the software.	4
Agility Degree:		24

Automatic Penetration Testing		Rating
Simplicity:	From our point of view, the analyst who executes the penetration testing tool must have knowledge about the vulnerabilities and attacks that the tool identified in order that he is capable of interpreting the results.	3
Free of Modeling and Documentation:	Same as in “Manual or semi-automatic Penetration Testing”.	4
Change Tolerate:	Full automatic tools use a fix set of attack pattern. They execute every time the same attacks until the set of attack pattern is updated. If new security requirements require testing new attack scenarios, which are not included in the set of the tool, these tools cannot test them. In case of testing a web application, if code has changed which is used on several sites, all these sites have to be tested again, because the modified or new code could have side effects.	3
Speed of Execution:	These tools can be run automatically over night or on weekends.	5
Informality:	By using such a kind of tool one knows whether the system is vulnerable to already known attacks.	3
People Orientation:	Since the work of this activity mainly consists of executing a tool it is completely method/tool based.	0
Iterative / Dividable	Tool could be used to attack specific entry points one by one.	5
Flexibility:	With this kind of tool one is capable of testing specific entry points whether they are vulnerable to publicly-known cyber-attack patterns.	4
Agility Degree:		27

Static Analysis		Rating
Simplicity:	Similar to Microsoft’s static analysis.	3
Free of Modeling and Documentation:	Similar to Microsoft’s static analysis.	3
Change Tolerate:	Similar to Microsoft’s static analysis.	5
Speed of Execution:	Similar to Microsoft’s static analysis.	5
Informality:	Similar to Microsoft’s static analysis.	3
People Orientation:	Similar to Microsoft’s static analysis.	3

Iterative / Dividable	Similar to Microsoft's static analysis.	5
Flexibility:	Similar to Microsoft's static analysis.	1
Agility Degree:		28

Red Team Testing		Rating
Simplicity:	Needs per definition security experts, which have the knowledge of hackers [81].	0
Free of Modeling and Documentation:	The main work consists of analyzing whether the system is secure by hacking the system. After hacking the system, the red team documents the identified vulnerabilities, successful attacks and not successful attacks which they conducted and report them to the management of the company.	4
Change Tolerate:	<p>The goal is to identify vulnerabilities in the software, to determine how they could be exploited and to analyse whether the organization recognize these attacks.</p> <p>If requirements are added or changed: In this case new attacks, which have not been considered before may be important to test now. We reckon that a new red team testing has to be performed to test these requirements.</p> <p>If requirements are removed: In this case successful attacks of the previous red team testing may be unnecessary, because requirements were tested that are not in place anymore. In our opinion, no new testing is needed.</p> <p>If code is modified: In this case the result of the performed attacks may not be as representative as before, because new vulnerabilities could have been inserted. The documentation of the successful attacks can be used to perform them again in order to test whether the detected vulnerabilities are fixed now. Furthermore, the documentation of the unsuccessful ones can also be used to execute them again in order to know if no new vulnerabilities were inserted so that attacks are now successful. However, in either way, new or modified code could always lead to new vulnerabilities. In order to ensure that the tested system component is still secure we think that a new red team testing has to be performed.</p> <p>Nevertheless, even if code has changed or new code has been written, or the requirements have changed, the management knows now if their employees are capable of handling such incidents.</p>	3
Speed of Execution:	Can last long, due to the fact that the red team may use, for example, social engineering, such as phishing. Furthermore, the organization's detection and response capabilities are tested. If it takes the employees of the company long to recognize the incident the speed of execution of this activity is obviously also long.	1
Informality:	By conducting this activity one knows if the system or network is vulnerable to predefined attacks. (Depends on the expertise of the red team, however they should be per definition high security experts)	5

People Orientation:	This activity is performed in team work, therefore in our opinion much people interaction is required.	4
Iterative / Dividable	Each member of the red team could attack the system on different entry points and each of them may conduct their own attacks hence, from our point of view, the work can be divided.	4
Flexibility:	The red team is rather flexible in testing the software. They plan themselves how they attack the system to exploit security flaws in the target application. For example, they can write their own exploitation code, can use pen tester tools to identify vulnerabilities in the target application, or they may use social engineering. With this kind of testing one can test every component of the system or network whether it is vulnerable to cyber-attacks.	5
Agility Degree:		26

Risk Based Testing		Rating
Simplicity:	We reckon that manual security testing requires much expertise in IT-security, because testers have to know several up-to-date attack patterns and should be capable of attacking an application.	2
Free of Modeling and Documentation:	This testing method needs some sort of risk assessment, which usually creates a high amount of documentation.	2
Change Tolerate:	The test-cases have to be prioritized by risk. Hence, this activity depends on a risk assessment. If requirements or code changes, a risk assessment must be revisited prior of retesting the functionality. In this risk assessment, new threats could appear and risks may be reprioritized. Based on this new knowledge it might be possible that new security controls have to be implemented and that software components have to be retested.	2
Speed of Execution:	-	2
Informality:	By conducting this activity one knows if the software is secure against several attack patterns. (Depends on the used attack patterns and the experience of the tester.)	4
People Orientation:	One or several person perform this activity manually. From our point of view, it is more people oriented than method-based.	4
Iterative / Dividable	The testing starts with test cases that have the highest risk and ends with the lowest one. The work of writing and executing test cases can be divided up among testers.	4
Flexibility:	Only high-risk functionality could be tested, due to the fact that the risks are known beforehand.	4
Agility Degree:		24

A.3 Security activities of SREP

Security Requirement Analysis		Rating
Simplicity:	Similar to Microsoft's Security Requirement Analysis.	2
Free of Modeling and Documentation:	In this activity nothing is implemented or tested.	0

Change Tolerate:	This activity depends on two activities, such as “Critical Assets” and “Risk Analysis” of SREP. Security requirements are derived from the results of these two activities.	2
Speed of Execution:	This analysis consists of several steps that can be seen in the “Iterative / Dividable” column below. However, This activity assumes that critical assets, likely attacker types and likely threats are already identified in other activities of SREP. This activity aims to derive security requirements from already existing documents in, for example, brain storming sessions with the customer. How these requirements are derived is not described in greater detail in [52].	4
Informality:	Without performing this activity the actual security needs of the customer are not known, such as security policies or security standards.	5
People Orientation:	The work consists of people who are brain storming about requirements, deriving them from already existing risk/threat models and documents, and prioritizing them.	4
Iterative / Dividable	This activity is performed in an iterative way. First, requirements are derived from, for example, legal requirements, then, further ones are derived from the results of the “Critical Assets” and “Risk Analysis” activity. Finally, the requirements are prioritized.	5
Flexibility:	Similar to Microsoft’s Security Requirement Analysis.	4
Agility Degree:		27

Agree on Definitions		Rating
Simplicity:	The goal of this activity is that the stake holders of the project and the company agrees on security policies, and ISO standards. Hence, to conduct this activity requires someone who has knowledge about several security ISO standards and policies.	2
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	According to our definition of change tolerance we measure what impact the change of requirements (functional- and security requirements) has on an activity. The aim of this activity is to agree on standards and policies. To comply to these security- standards and policies is some kind of requirement.	3
Speed of Execution:	-	2
Informality:	By conducting this activity one knows which standards and security definitions should be complied through the lifetime of the project.	5
People Orientation:	The work consists mainly on discussing with the customer and stakeholders about the security standards and policies.	4

Iterative / Dividable	This activity is usually performed at the beginning of a project and not revisited in later phases. However, agile development revisits the inception or analysis phase several times in the lifetime of a project. In agile development processes not all requirements are known when the project starts, therefore this activity has to be performed several times, because it could happen that new functionalities require new security policies and additional security visions. Hence, this activity should be performed in an iterative manner, because of the iterative nature of agile development processes. From our point of view, it should be performed in the context of an epic or several epics, which are belonging together. When epics are defined, the functionality, which should be implemented, is already known and security policies can be defined. They can then be refined in the context of user stories. In this way, they are defined when the functionality is actually known and therefore early in the development process. In each iteration, the work can heavily be divided into subtasks, because the work consists only of defining the stake holders and on agreeing on standards and policies. Hence, we give this activity only four points instead of five in this <i>agile feature</i> .	4
Flexibility:	Which security- policies, and standards should be complied throughout the lifecycle of a software project depend, in our opinion, on the wishes of the stake holders and do not allow much tolerance.	1
Agility Degree:		21

Risk Analyses (CRAMM)		Rating
Simplicity:	Most of the work consists only of asking structured questions to employees, for example, asset owners, or employees with security expertise in different categories. The answers are then entered in the program and based on the answers the tool presents risks and security controls that can be used to prevent these risks. We reckon that the analyst should have basic knowledge about IT-security in order to guide employees on answering the questions and to explain the interviewees some security terms.	2
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerance:	If security- or functional requirements change new assets may be added, modified or removed. Therefore, we reckon that a new analysis has to be conducted and several questions of the tool should be asked again.	3
Speed of Execution:	Several meetings have to be held. We think that it could be quite complicated to arrange meetings with all the participants. This could take some time.	2
Informality:	By conducting this activity one knows vulnerabilities and threats of an application and how to mitigate these threats on a rather high-level. However, this information is only based on the answers of several employees which have not to be security experts.	3
People Orientation:	Risks are calculated with the help of a tool, nevertheless, many people are involved in this analysis, because several interviews are held.	4
Iterative / Dividable	This methodology consists of several steps.	5

Flexibility:	The tool has a strict set of questions. Based on the answers to these questions it calculates risks and suggests security controls to prevent these risks. Therefore, the analysts have to strictly ask the questions which the tool provides.	1
Agility Degree:		20

Critical Assets		Rating
Simplicity:	To identify assets which are vulnerable to attacks we reckon that the analyst should have domain specific knowledge and has obviously to know the system. An employee of the management should also be involved in order to rate the impact on loss of an asset.	3
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	If requirements change assets could be modified, removed are new ones added. In our opinion, in the context of a user story only several assets may be modified, removed are new ones added. The removing or modifying of assets from the critical assets list is not time intensive, because they are already known. However, the identification of new ones, because some functionality was added may takes some time.	4
Speed of Execution:	This is, in our opinion, a lightweight activity and in the context of a user story only few assets can be added or modified, thus we think that the analysis is quick.	4
Informality:	By conducting this activity one knows what could be of value for an attacker.	4
People Orientation:	The work mainly consists of identifying assets with methods, such as interviews with stakeholders, and selecting them with the help of lists of assets.	4
Iterative / Dividable	In each user story new assets may be added, modified, or removed. Hence, it does make sense to perform this activity for each user story.	5
Flexibility:	In our opinion, it is important to identify as much assets possible in order to know what could be of value for an attacker.	1
Agility Degree:		25

Identify Threats and Develop Artifacts		Rating
Simplicity:	The aim of this activity is to develop artifacts, such as UMLsec diagrams, attack trees, or abuse cases of an application and to derive threats from them. Therefore, the one who conducts this activity should be aware of some of these modeling techniques and should know how threats can be derived from these artifacts.	1
Free of Modeling and Documentation:	The creation of artifacts produces much documentation.	0
Change Tolerate:	The change of functional requirements or code may require updating, diagrams, abuse cases or attack trees.	2
Speed of Execution:	-	3
Informality:	By conducting this activity one knows the threats, which could arise when a user story is implemented. (Depends on the knowledge of the analyst)	4
People Orientation:	To develop artifacts, such as UMLsec diagrams is, in our opinion, method-based.	1

Iterative / Dividable	UMLSec diagrams and sequence diagrams can be modelled for specific software components and be updated if new functionality is added in new user stories. For each new software increment, attack trees or abuse cases can be modelled to illustrate how this new functionality can be abused.	4
Flexibility:	The one who conducts this activity is rather flexible in deciding which artifact he creates, because the authors of [52] did not specify which specific technique should be used, but rather listed several which could be used. For the identification of threats only threats which could harm critical assets, which were identified in the "Critical Assets" activity could be considered.	3
Agility Degree:		18

Requirement Inspection		Rating
Simplicity:	The analyst should be familiar in using EALs and assurance requirements of the Common Criteria, and SSE-CMM in order to assure that the defined security requirements take all identified threats into account, that all security objectives are satisfied, and each security requirement is met.	2
Free of Modeling and Documentation:	This activity aims to verify that the application developed is secure. SSE-CMM and the EALs of the Common Criteria are used to conform that all requirements are met. The verification process has to be documented in a report, which produces much documentation.	3
Change Tolerate:	If security requirements change or new ones are added it must be verified with Common Criteria whether the application meet these requirements. If functional requirements change it may be necessary to update several documents, such as UMLSec diagrams. These diagrams have also to be reviewed again.	3
Speed of Execution:	In our opinion, it could take some time to review all requirements and produced documents. We reckon that the most time intensive part is to assure with the help of EALs of the Common Criteria and the PAs of the SSE-CMM whether all threats are countered, security policies are enforced, and assumptions are upheld.	3
Informality:	By conducting this activity one knows if all steps of SREP have been performed properly and whether all security requirements are complete and each security objective is achieved.	4
People Orientation:	In our opinion, verifying an application with Common Criteria and SSE-CMM to ensure that all defined security requirements are met and reviewing all produced documents and artifacts is rather method-based.	2
Iterative / Dividable	Scrum produces iteratively software increments and artifacts. By integrating security activities, security artifacts are also developed iteratively, which could be reviewed.	5
Flexibility:	The work consists of reviewing all produced artifacts from previous steps, which obviously means that all artifacts must be reviewed. The methods which are used for the verification are strictly method based and do not allow much flexibility.	0
Agility Degree:		22

Repository Improvement	Rating
-------------------------------	--------

Simplicity:	The one who conducts this activity has to know which threats are likely to occur in further development. In our opinion, in order to achieve this in an effective way, the analyst should be aware of what functionalities are already in discussion with the customer, and should frequently review several security test and code review results.	3
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	The intent of this activity is to document all artifacts which were developed by conducting activities of SREP, which may be needed for further developments. In our opinion, the change of requirements has no impact on this activity.	5
Speed of Execution:	The artifacts have already been already created. Only the ones which could be of use in further development should be chosen. In our opinion, this does not requirement much time.	5
Informality:	These selected artifacts can speed up further threat analysis.	3
People Orientation:	The work consists of documenting threats which could occur in further development.	1
Iterative / Dividable	Threats are added iteratively into a database.	5
Flexibility:	From our point of view, it is hard to decide, which artifacts are needed in the future, because no one can divine the future. Therefore, the one who performs this activity has much flexibility in deciding which should be documented.	3
Agility Degree:		25

A.4 Security activities of NIST 800-64

Initiate Project Security Planning		Rating
Simplicity:	We think that for conducting this activity one has to know where to look for security sources and has to understand them. One has to understand the security objectives, requirements and has to know which tools and technologies should be used. The analyst has also to have some basic security knowledge in order that he can explain security implications to stake holders.	1
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	The aim of this activity is to initiate the project security planning. We reckon that it does not make sense to perform it for every user story, because it should obviously one be performed when a project is initiated. Our definition of change tolerance is that the activity should be tolerant to small changes in requirements and therefore not much additional work should be required on small requirement changes. The security requirements are in this activity defined on a rather high-level, because the focus of this activity is to give the stakeholders an initial overview of the complexity and impacts. In our opinion, on small security requirements changes this activity should not be performed again. However, on dramatic requirement changes it may be necessary to update the current security vision and security milestones.	4

Speed of Execution:	The identification of high-level security requirements, stake holders, and tools and technologies which should be used throughout the lifecycle of the software development could take some time. However, if they are chosen at the beginning of an agile project we think that in later epics and user stories only several changes are needed.	5
Informality:	Standards and policies are defined and a high-level overview of the security requirements and the key milestones are given.	5
People Orientation:	Many people are involved, because we think stake holders, security experts, software architects, and management personnel should be involved in defining the security policies, standards, requirements, used tools and security key milestones. The aim of this activity is to give all stakeholders a basic overview of the security objectives, requirements, used tools, and technologies.	5
Iterative / Dividable	In Scrum software is implemented in increments and therefore the identification of security milestones and security requirements should also be performed iteratively. However, in our opinion, it does not make sense to perform this activity for every user story, but rather for several epics, which are belonging together and implement a certain software component.	5
Flexibility:	The security requirements and key milestones depend on the wishes of the customer. From our point of view, the used tools or technologies and security roles can be chosen freely by the company and therefore the company is rather flexible in choosing them.	3
Agility Degree:		28

Categorize Information System		Rating
Simplicity:	The authors of [47] stated that personnel with responsibilities on security management, with interest on assets, such as information owners, developers, project managers and also employee with responsibilities in security implementations should be involved in this activity.	2
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	Similar to Critical Assets.	4
Speed of Execution:	Similar to Critical Assets.	4
Informality:	By conducting this activity one knows what could be of value for an attacker and also knows the application's impact on availability, integrity and confidentiality.	4
People Orientation:	The authors of [92] provide much support in form of lists, metrics and examples for this activity, therefore we reckon that an essential part of the work is method based. However, as stated in the "Simplicity" column several roles are involved in performing this activity. They have to identify the information types and also have to discuss which impact levels should be assigned for each of them, because these impact levels are not defined in great detail and allow in our opinion much room for debating. Furthermore, the provisional impact levels are reviewed, which we think is quite people-oriented.	3
Iterative / Dividable	In every user story assets or information systems could be added or modified, therefore, in our opinion, this activity can be performed every Sprint.	4
Flexibility:	Similar to Critical Assets.	1

Agility Degree: 22

Assess Business Impact		Rating
Simplicity:	According to [93, p. 15] a person who manages resources, such as server hardware should perform this activity with the help of management personnel. Hence, no employee with security expertise is needed for this activity.	5
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	The availability impact on the information system from the “Categorize Information System” activity have to be considered here, therefore it depends on this activity. Requirement changes may add, remove or modify business processes, which have then to be considered on this activity.	2
Speed of Execution:	Identification of business processes, impact analysis, identification of recovery strategies, resource identification, and cost balancing needs in our opinion some time even in the context of a single user story.	3
Informality:	By conducting this activity one knows the impact of business processes on availability, integrity and confidentiality and also knows the costs of specific downtimes, which resources are needed to recover, and is aware of several backup strategies.	4
People Orientation:	This activity is, from our point of view, rather method based, because cost analysis, down time calculations have to be conducted.	1
Iterative / Dividable	This activity should be revisited periodically if, for example, new resources are added or modified [93, p. 17].	4
Flexibility:	Only high-security critical business processes could be considered in this activity. However, beforehand the high-critical ones are not known. The aim of this activity is to identify them.	1
Agility Degree:		21

Assess Privacy Impact		Rating
Simplicity:	The aim of this activity is to determine which components of the application stores, transmit or creates sensitive information. From our point of view, domain specific knowledge and basic security knowledge is required to complete this task.	3
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	If new functionality is added it has to be reviewed whether it processes, or stores private information. However, in our opinion, if security requirement changes that does not have any impact on whether software components store, or process private information.	4
Speed of Execution:	We reckon that this activity is rather light-weight, because only questions to address privacy information of information systems are asked. The selection of proper security controls is performed in other activities of this process.	4
Informality:	By conducting this activity one knows which system processes private information.	4

People Orientation:	The analysts ask questions, such as “Why is the information being collected?”, “With whom will the information be shared?”, “How will the information be secured?” and document the answers. Therefore, the main work of this activity is to ask questions with respect to whether the system processes privacy information, which could be done with the help of several other people in brain storming sessions. Therefore, we reckon that most of the work is more people-oriented than method-based.	4
Iterative / Dividable	As described in the row above, the main work consists of asking questions. This work can be, from our point of view, divided among several person.	5
Flexibility:	We reckon that in order that this activity is effective a privacy assessment has to be performed for each information system where it is not obvious that no privacy information is transmitted, stored or created. In our opinion, it is important to be familiar with which software components create, store, and process private information, because this information helps to define security requirements.	1
Agility Degree:		25

The "Assess Risk to System" activity is described in NIST SP 800-30 [64]. The authors of this document described this activity on a rather high-level and did not state, for example, how threats or vulnerabilities can be identified in detail. They only provide several tables which list, for example, input sources for identifying threats, and vulnerabilities, a table which states various threat-, impact-, and predisposing condition types. Furthermore, they provide a table that lists various threat events, and several metrics for assessment scales for evaluating, for example, the impact of an threat event, the pervasiveness of predisposing conditions, and the likelihood of threat events. However, they stated that these metrics and tables are not mandatory for this method. For analyzing this activity in regard of how agile it is we assume that all steps described in Section 5.2.4.5 are conducted in brain storming sessions with the support of the provided tables and metrics.

Assess Risk to System		Rating
Simplicity:	This activity aims to perform a risk assessment without tool support, hence, according to our metric it is complex.	2
Free of Modeling and Documentation:	Nothing is implemented or tested.	0
Change Tolerate:	Similar to Threat Modeling.	2
Speed of Execution:	-	2
Informality:	By conducting this activity one knows the risks, threats, and vulnerabilities of an information system. However, this depends on the exact method used for identifying threats and vulnerabilities and the expertise of the one who conducts it.	4

People Orientation:	<p>This methodology consists of several steps. The steps that are used throughout the method are: (1) Evaluating organizational risks with the help of provided metrics for assessing various scales for, for example, the likelihood of that a threat event results in adverse impacts and the possible impact of a threat event. (2) In brain storming sessions threats, vulnerabilities and predisposing conditions are assessed with the help of tables which list several threat sources-, vulnerability-, and predisposing types. Additionally, to support the identification of threat events the NIST SP 800-30 provides several examples.</p> <p>In our opinion, the first point is rather method based whereas the second point requires people interactions, due to brain storming sessions are conducted. We reckon that it is more method-based than people oriented, because the goal of this assessment is to determine organizational risks, which depends on several risk factors that have to be graded either qualitative, quantitative, or semi-qualitative.</p>	2
Iterative / Dividable	<p>This activity consists of several steps, hence it is dividable. The authors of NIST SP 800-30 state that this method should be performed iteratively, first on a high-level and then in greater detail in later phases of the development lifecycle. However, in our opinion, this activity is quite heavy-weight, due to many steps have to be performed. Therefore, we only give it four points instead of five in this <i>agile feature</i>.</p>	4
Flexibility:	<p>From our point of view, in order to assess organizational risks all steps described in the "People Orientation" row have to be performed, due to they depend on each other. For example, risk cannot be determined if the likelihood of that an adversary threat successfully exploits a vulnerability or the scale of the impact of this threat is missing. Obviously, for these grades threats have to be assessed first, which on the other hand require the identification of threat sources which initiate threat events. Next, also vulnerabilities and predisposing conditions in which the vulnerability that a threat exploits which finally produces risk must be analysed. However, the authors of [64] did not describe a strict way in how these steps and to which degree they have to be performed and therefore provide much flexibility. For example, only obvious high-risky threat sources and threat events could be considered.</p>	3
Agility Degree:		19

Select and Document Security Controls		Rating
Simplicity:	<p>The authors of [63, p. 24–27] suggest that amongst other supporting roles with low security expertise someone who has expertise in making risk-related and organization-wide decisions should perform this activity. Due to the fact that the controls are selected from catalogs, in our opinion, the complexity is not as great as for example in designing secure architectures.</p>	2
Free of Modeling and Documentation:	<p>Nothing is implemented or tested.</p>	0

Change Tolerate:	<p>If requirements change it might be necessary to perform a so called gap analysis [65, p. 44–45], which consists of the following steps:</p> <ul style="list-style-type: none"> • Update the security category of the information system, • Review the existing security plan, • Reassess the risk, • Select further security controls if needed, • Implement the security controls <p>This activity depends therefore on the results of other activities of the NIST 800-64 process, such as “Categorize Information System” and “Assess Risk to System”. If changes occur these activities must be revisited first, thus much additional effort is needed.</p>	1
Speed of Execution:	-	3
Informality:	By conducting this activity one knows which security controls can be used to reduce or prevent risks.	4
People Orientation:	The main work consists of selecting basic security controls of a predefined catalog, to tailor them, and to select further security controls which reduce or prevent risks identified from risk assessments. In our opinion, the selection of basic security controls and corresponding more specific security controls from catalogs is method-based. However, the tailoring process and the selection of more security controls for reducing or preventing risks of risk assessment requires, from our point of view, brain storming sessions, which requires people interactions.	2
Iterative / Dividable	The process consists of several steps, which are performed iteratively and are, from our point of view, quite light-weight.	5
Flexibility:	In our opinion, all steps of this activity should be performed to be effective, because without selecting and implementing security controls a security process does, from our point of view, not make any sense. However, we reckon that the company has much flexibility in tailoring the selected base security controls and in deciding whether security controls are necessary for specific cases.	3
Agility Degree:		20

Design Security Architecture		Rating
Simplicity:	From our point of view, much security expertise is necessary to design secure software architectures.	1
Free of Modeling and Documentation:	Nothing is implemented or tested.	0

Change Tolerate:	<p>In our opinion, the change tolerance of this activity depends on whether the security architecture is designed for a legacy system or a new development.</p> <p>For new development (no code has been implemented yet) For a new development the architecture can be changed, for example, if new risks were identified, new security controls were selected, or requirements have changed. In this case only the documentation has to be revisited and obviously no code has to be changed. This activity depends therefore on the “Select and document security controls” activity.</p> <p>For legacy systems (code has been already implemented) In this case, we reckon that the change tolerance depends on the designed security architecture. If only the minimal security requirements were considered for the design a lot of effort may be needed to redesign the system. If requirements change in a way that the architecture has to be changed, obviously not only the documentation has to be revisited, but also steps such as, identifying new security controls and / or modifying and implementing them. Hence, this activity depends on the “Select and document security controls” activity and has an impact on the implementation of security controls.</p>	1
Speed of Execution:	We suggest that this activity should not be performed in the context of a single user story, but rather in the context of an epic or several epics, which are belonging together. For simplicity we say that several epics which are belonging together are a big epic. In this way, the security architect has a bigger view of the functionality which should be implemented. With this bigger view he is capable of zoning the functionality of the separate user stories together or distribute them. He can also consider common external dependencies for these functionalities and can design epic-wide or even big epic-wide security controls. We assume for this analysis that the requirements are known and that security controls were already selected in other activities.	5
Informality:	By conducting this activity one knows a secure architecture of the system.	4
People Orientation:	Similar to "Design Requirements" activity of Microsoft SDL.	3
Iterative / Dividable	In Scrum, software is developed in increments; therefore, we reckon that the security architecture has also to be developed in increments. If a new functionality has to be implemented the already existing security architecture could be adopted and several security controls could be reused.	4
Flexibility:	The authors of [47, p. 24] stated that this activity provides the most value for the system in regard of costs, therefore we reckon that it should be performed thoroughly. However, the designers have much flexibility in deciding how they design the software, because security flaws can be prevented in more than one way.	2
Agility Degree:		20

Simplicity:	The assessment methods are selected with the help of a catalog. Furthermore, best practice assessment cases are also provided. We reckon that for this step not much security expertise is needed, due to the fact that controls are only selected from a catalog. However, for performing the individual security testing techniques we think that security expertise is required. NIST SP 800-53A defined this activity very general, because they describe various testing techniques and tools which could be used for testing specific security controls. The complexity depends on the individual testing method which is used. For example, manual penetration testing requires hacking skills, whereas automatic “all-in-one” penetration testing tools do not require much security expertise.	1-2
Free of Modeling and Documentation:	This activity mainly consists of testing the system, but it is also necessary to document the results.	4
Change Tolerate:	<p>The authors of [47, p. 27] stated that based on the results of this activity other security activities of this process may require to be revisited. It may require changing security requirements, modifying of the security architecture, and it may also require updating of risk assessments in order to reflect the current security controls.</p> <p>According to the authors of [66, p. 14], a risk assessment has to be performed prior to this activity and obviously the assessment methods can only be chosen from the assessment catalog when the security controls were identified. Thus, we reckon that if requirements change it may be needed to first update the risk assessment, to re-select security controls, and to update the secure design. Only after all these steps have been performed, safeguards are implemented and can be tested.</p>	0-1
Speed of Execution:	We reckon that the speed of execution largely depends on the chosen testing method, as mentioned in the “Simplicity” row.	2-5
Informality:	The goal of this activity is to assure that all implemented security controls meet the security requirements. However, to which degree it can be verified whether the defined security requirements are met depends on the concrete chosen testing method.	4
People Orientation:	As stated in the row below testing should be automated as much as possible. The use of automated testing tools is purely method-based, whereas the writing of, for example, unit or integration tests requires people interaction, due to the fact that developers have to write these test-cases first.	3
Iterative / Dividable	According to the authors of [47, p. 26–27] this process should focus on specificity, repeatability, and iteration. This means that the software should be tested in the environment in which the system is later deployed, the tests must be able to be performed several times, and should deliver successful results multiple times. Thus, we give this activity five points for being iterative.	5
Flexibility:	On the one hand, in this activity, security assessment procedures should be selected from a predefined security assessment procedure catalog for assessing security controls, which does, from our point of view, not provide much flexibility. On the other hand, we reckon that there is much flexibility in deciding which depth or coverage degree the organization chooses for assessing security controls.	3
Agility Degree:		22-27

Bibliography

- [1] Acunetix Web Vulnerability Scanner. <https://www.acunetix.com/vulnerability-scanner/> (retrieved on August 7, 2016) (cited on page 33).
- [2] Agile Alliance. *Definition of Ready*. <https://www.agilealliance.org/glossary/definition-of-ready/> (retrieved on August 7, 2016) (cited on page 18).
- [3] Agile Alliance. *Invest*. <https://www.agilealliance.org/glossary/invest/> (retrieved on August 7, 2016) (cited on page 18).
- [4] Agile Alliance. *User Stories*. <https://www.agilealliance.org/glossary/user-stories/> (retrieved on August 7, 2016) (cited on page 18).
- [5] Agile Alliance. *Agile manifesto*. 2001. <http://www.agilemanifesto.org> (retrieved on October 22, 2016) (cited on pages 5, 15, 52).
- [6] Agile Alliance. *Principles behind the Agile Manifesto*. 2001. <http://agilemanifesto.org/principles.html> (retrieved on October 22, 2016) (cited on page 15).
- [7] Ståle Amland. “Risk-based testing:: Risk analysis fundamentals and metrics for software testing including a financial application case study”. In: *Journal of Systems and Software* 53.3 (2000), pages 287–295. ISSN 01641212. doi:[http://dx.doi.org/10.1016/S0164-1212\(00\)00019-4](http://dx.doi.org/10.1016/S0164-1212(00)00019-4). <http://www.sciencedirect.com/science/article/pii/S0164121200000194> (cited on page 58).
- [8] Cynthia Andres and Kent Beck. *Extreme Programming Explained: Embrace Change*. 2nd edition. Addison-Wesley Professional, 2004. ISBN 0321278658 (cited on pages 5, 9, 10, 15, 47).
- [9] Vähä-Sipilä Antti. “Risk management”. In: *Handbook of the Secure Agile Software Development Life Cycle*. Edited by Pietikäinen Pekka and Röning Juha. Oulu: University of Oulu, 2014. Chapter 5, pages 29–43 (cited on pages 2, 13).
- [10] Vähä-Sipilä Antti. “Security in Agile Product Management”. In: *Handbook of the Secure Agile Software Development Life Cycle*. Edited by Pietikäinen Pekka and Röning Juha. Oulu: University of Oulu, 2014. Chapter 3, pages 15–21 (cited on pages 2, 12).
- [11] Vishal Asthana et al. *Practical Security Stories and Security Tasks for Agile Development Environments*. 2012. http://www.safecode.org/publication/SAFECode_Agile_Dev_Security0712.pdf (retrieved on April 5, 2016) (cited on pages 2, 6, 10, 11, 43, 44, 82).
- [12] Tigist Ayalew, Tigist Kidane, and Bengt Carlsson. “Identification and Evaluation of Security Activities in Agile Projects”. In: *Secure IT Systems: 18th Nordic Conference, NordSec 2013*. (Ilulissat, Greenland). Edited by Hanne Riis Nielson and Dieter Gollmann. Volume 8208. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, October 18, 2013, pages 139–153. ISBN 9783642414886. doi:10.1007/978-3-642-41488-6_10. http://dx.doi.org/10.1007/978-3-642-41488-6_10 (cited on page 42).

- [13] Dejan Baca and Bengt Carlsson. “Agile Development with Security Engineering Activities”. In: *Proceedings of the 2011 International Conference on Software and Systems Process*. (Waikiki, Honolulu, HI, USA). ICSSP '11. New York, NY, USA: ACM, 2011, pages 149–158. ISBN 9781450307307. doi:10.1145/1987875.1987900. <http://doi.acm.org/10.1145/1987875.1987900> (cited on pages 2, 5, 11, 42, 53, 63).
- [14] Dejan Baca and Kai Petersen. “Countermeasure graphs for software security risk assessment: An action research”. In: *Journal of Systems and Software* 86.9 (2013), pages 2411–2428 (cited on page 4).
- [15] Jason Bau et al. “State of the Art: Automated Black-Box Web Application Vulnerability Testing”. In: *2010 IEEE Symposium on Security and Privacy*. May 2010, pages 332–345. doi:10.1109/SP.2010.27 (cited on page 33).
- [16] M Belk et al. *Fundamental Practices for Secure Software Development 2nd Edition: A Guide to the Most Effective Secure Development Practices in Use Today*. Edited by Stacy Simpson and SAFECode. Software Assurance Forum for Excellence in Code (SAFECode). February 8, 2011. https://www.safecode.org/publication/SAFECode_Dev_Practices0211.pdf (cited on page 82).
- [17] Scott Berkun. *Making Things Happen: Mastering Project Management*. “O’Reilly Media”, July 2008. ISBN 0596523335 (cited on page 5).
- [18] Konstantin Beznosov and Philippe Kruchten. “Towards Agile Security Assurance”. In: *Proceedings of the 2004 Workshop on New Security Paradigms*. (Nova Scotia, Canada). NSPW '04. New York, NY, USA: ACM, 2004, pages 47–54. ISBN 1595930760. doi:10.1145/1065907.1066034. <http://doi.acm.org/10.1145/1065907.1066034> (cited on pages 2, 33, 46, 47).
- [19] Barry Boehm and Richard Turner. “Management challenges to implementing agile processes in traditional development organizations”. In: *IEEE software* 22.5 (2005), pages 30–39 (cited on page 1).
- [20] Marting Buchi and Wolfgang Weck. “The Greybox Approach: When Blackbox Specifications Hide Too Much”. In: *TUCS Technical Report No 297a (Revised)* (August 1999) (cited on page 33).
- [21] Pravir Chandra. *Software Assurance Maturity Model - A guide to building security into software development - version 1.1*. 2016. https://www.owasp.org/images/d/d8/OpenSAMM_Core_V1-1-Final.pdf (retrieved on July 5, 2016) (cited on pages 2, 7, 111).
- [22] Checkmarx. *Static Code Analysis (SAST)*. 2016. <https://www.checkmarx.com/technology/static-code-analysis-sca/> (retrieved on July 5, 2016) (cited on page 71).
- [23] Brian Chess and Gary McGraw. “Static analysis for security”. In: *IEEE Security & Privacy* 2.6 (November 2004), pages 76–79. ISSN 1540-7993. doi:10.1109/MSP.2004.111 (cited on page 34).
- [24] The MITRE Corporation. *Common weakness enumeration*. 2016. <https://cwe.mitre.org/> (retrieved on October 11, 2016) (cited on pages 6, 43).
- [25] The MITRE Corporation. *CWE-862: Missing Authorization*. 2016. <https://cwe.mitre.org/data/definitions/862.html> (retrieved on October 11, 2016) (cited on page 83).
- [26] Robert Crook et al. “Security requirements engineering: When anti-requirements hit the fan”. In: *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*. IEEE Computer Society. 2002, pages 203–205. doi:10.1109/ICRE.2002.1048527 (cited on pages 2, 43).
- [27] Adam Doupé, Marco Cova, and Giovanni Vigna. “Why Johnny Can’t Pentest: An Analysis of Black-Box Web Vulnerability Scanners”. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Edited by Christian Kreibich and Marko Jahnke. Berlin, Heidelberg: Springer

- Berlin Heidelberg, July 2010, pages 111–131. ISBN 9783642142154. doi:10.1007/978-3-642-14215-4_7. http://dx.doi.org/10.1007/978-3-642-14215-4_7 (cited on page 33).
- [28] Pär Emanuelsson and Ulf Nilsson. “A comparative study of industrial static analysis tools”. In: volume 217. Elsevier, 2008, pages 5–21. doi:<http://dx.doi.org/10.1016/j.entcs.2008.06.039>. <http://www.sciencedirect.com/science/article/pii/S1571066108003824> (cited on page 133).
- [29] Xiaocheng Ge et al. “Agile Development of Secure Web Applications”. In: *Proceedings of the 6th International Conference on Web Engineering*. (Palo Alto, California, USA). ICWE '06. New York, NY, USA: ACM, 2006, pages 305–312. ISBN 1595933522. doi:10.1145/1145581.1145641. <http://doi.acm.org/10.1145/1145581.1145641> (cited on pages 2, 13).
- [30] Asif Qumer Gill and Brian Henderson-Sellers. “Comparative evaluation of XP and scrum using the 4d analytical tool (4-DAT)”. In: *Proceedings of the European and Mediterranean conference on information systems*. (Costa Blanca, Spain). EMCIS, July 6, 2006, pages 1–8. <https://opus.lib.uts.edu.au/bitstream/10453/6967/1/2006005499.pdf> (cited on pages 10, 43).
- [31] Paolo Giorgini et al. “Modeling security requirements through ownership, permission and delegation”. In: *13th IEEE International Conference on Requirements Engineering (RE'05)*. IEEE Computer Society, August 2005, pages 167–176. doi:10.1109/RE.2005.43 (cited on pages 2, 43).
- [32] Anjana Gosain and Ganga Sharma. “A Survey of Dynamic Program Analysis Techniques and Tools”. In: *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014: Volume 1*. Edited by Suresh Chandra Satapathy et al. Cham: Springer International Publishing, 2015, pages 113–122. ISBN 9783319119335. doi:10.1007/978-3-319-11933-5_13. http://dx.doi.org/10.1007/978-3-319-11933-5_13 (cited on page 34).
- [33] Jim Highsmith. “Agile project management: Principles and tools”. In: *Agile Project Management Advisory Service Executive Report* (2003) (cited on page 1).
- [34] Michael Howard and Steve Lipner. *The security development lifecycle: SDL, a process for developing demonstrably more secure software*. Microsoft Press, 2006. ISBN 0735622140 (cited on pages 1, 6, 22, 37, 51, 56, 57, 80, 129, 133, 134).
- [35] *IBM Security AppScan*. <http://www-03.ibm.com/software/products/de/appscan> (retrieved on August 7, 2016) (cited on page 33).
- [36] *Information technology – Security techniques – Evaluation criteria for IT security – Part 1: Introduction and general model*. ISO/IEC 15408-1:2009. International Organization for Standardization, December 2009 (cited on page 28).
- [37] *Information technology – Security techniques – Evaluation criteria for IT security – Part 2: Security functional components*. ISO/IEC 15408-2:2008. International Organization for Standardization, August 2008 (cited on page 28).
- [38] *Information technology – Security techniques – Evaluation criteria for IT security – Part 3: Security assurance components*. ISO/IEC 15408-3:2008. International Organization for Standardization, August 2008 (cited on page 28).
- [39] *Information technology – Security techniques – Systems Security Engineering – Capability Maturity Model (SSE-CMM)*. ISO/IEC 21827:2008. International Organization for Standardization, October 2008 (cited on pages 2, 20, 28–32, 46).
- [40] *Information technology – Security techniques – Information security risk management*. ISO/IEC 27005:2011. International Organization for Standardization, June 2011 (cited on page 26).

- [41] *Information technology – Guidelines for the management of IT Security – Part 1: Concepts and models for IT Security*. ISO/IEC TR 13335-1:1996. International Organization for Standardization, December 1996 (cited on pages 19, 20).
- [42] M. Tim Jones. *Static and dynamic testing in the software development life cycle*. August 26, 2013. <https://www.ibm.com/developerworks/library/se-static/> (retrieved on August 7, 2016) (cited on page 34).
- [43] Paul C. Jorgensen. *Software Testing: A Craftsman’s Approach*. 1st edition. Boca Raton, FL, USA: CRC Press, Inc., 1995. ISBN 084937345X (cited on page 33).
- [44] Sathya Prakash Kadhivelan and Andrew Söderberg-Rivkin. “Threat Modelling and Risk Assessment”. In: (2014) (cited on page 42).
- [45] Feisal Keblawi and Dick Sullivan. “Applying the common criteria in systems engineering”. In: *IEEE Security & Privacy* 4.2 (March 2006), pages 50–55. ISSN 1540-7993. doi:10.1109/MSP.2006.35 (cited on page 1).
- [46] Hossein Keramati and Seyed-Hassan Mirian-Hosseinabadi. “Integrating software development security activities with agile methodologies”. In: *2008 IEEE/ACS International Conference on Computer Systems and Applications*. IEEE Computer Society, March 2008, pages 749–754. doi:10.1109/AICCSA.2008.4493611 (cited on pages 2–5, 11, 12, 51, 52, 121).
- [47] Richard Kissel et al. “Security Considerations in the System Development Life Cycle”. In: *NIST SP 800-64 revision 2* (2008). doi:10.6028/NIST.SP.800-64r2 (cited on pages 1, 35, 36, 51, 60–62, 69, 146, 151, 152).
- [48] Panagiotis Louridas. “Static code analysis”. In: *IEEE Software* 23.4 (2006-07), pages 58–61. ISSN 0740-7459. doi:10.1109/MS.2006.114 (cited on pages 34, 133).
- [49] Zhendong Ma, Friedrich Kupzog, and Murdock Paul. “Secure Development Life Cycle”. In: *Smart Grid Security: Innovative Solutions for a Modernized Grid*. Syngress, August 12, 2015, pages 219–245. ISBN 0128021225 (cited on pages 33, 47, 137).
- [50] Gary McGraw. *Software Security: Building Security In*. Volume 1. Addison-Wesley Professional, 2006. ISBN 0321356705 (cited on pages 1, 39, 51, 57, 58, 93, 137).
- [51] J.D. Meier et al. *Improving Web Application Security: Threats and Countermeasures*. Microsoft Press, September 2003. ISBN 0735618429 (cited on page 21).
- [52] Daniel Mellado, Eduardo Fernández-Medina, and Mario Piattini. “A Common Criteria Based Security Requirements Engineering Process for the Development of Secure Information Systems”. In: *Computer Standards & Interfaces* 29.2 (February 2007), pages 244–253. ISSN 0920-5489. doi:10.1016/j.csi.2006.04.002 (cited on pages 4, 38, 51, 58, 59, 70, 86, 141, 144).
- [53] *Metasploit*. <https://www.metasploit.com/> (retrieved on August 7, 2016) (cited on pages 34, 47).
- [54] Microsoft. *Microsoft Threat Modeling Tool 2016*. <https://www.microsoft.com/en-us/download/details.aspx?id=49168> (retrieved on August 7, 2016) (cited on page 21).
- [55] Microsoft. *Microsoft Application Verifier*. 2008. <https://www.microsoft.com/en-us/download/details.aspx?id=20028> (retrieved on October 11, 2016) (cited on page 34).
- [56] Microsoft. *Microsoft: Security Development Lifecycle - SDL Process Guidance Version 5.2*. May 23, 2012. <https://www.microsoft.com/en-us/download/details.aspx?id=29884> (retrieved on April 5, 2016) (cited on page 37).
- [57] Microsoft. *SDL for Agile*. 2012. <https://www.microsoft.com/en-us/SDL/Discover/sdlagile.aspx> (retrieved on August 7, 2016) (cited on pages 2, 3, 12, 87).

- [58] Microsoft. *Microsoft Threat Modeling Tool 2016 User Guide*. 2015. <https://www.microsoft.com/en-us/download/details.aspx?id=49168> (retrieved on January 1, 2017) (cited on page 22).
- [59] Suvda Myagmar, Adam J Lee, and William Yurcik. “Threat modeling as a basis for security requirements”. In: *Symposium on requirements engineering for information security (SREIS)*. Volume 2005. Citeseer. 2005, pages 1–8 (cited on pages 2, 43).
- [60] Srinivas Nidhra and Jagruthi Dondeti. “Blackbox and whitebox testing techniques-a literature review”. In: *International Journal of Embedded Systems and Applications (IJESA)* 2.2 (June 2012), pages 29–50. doi:10.5121/ijesa.2012.2204 (cited on page 33).
- [61] NIST. *Assessment Cases - Download Page*. <http://csrc.nist.gov/groups/SMA/fisma/assessment-cases.html> (retrieved on January 1, 2017) (cited on page 63).
- [62] NIST. *Privacy Impact Assessment (PIA)*. <https://www.nist.gov/sites/default/files/documents/director/oism/NIST-TIP-PIA-Consolidated.pdf> (retrieved on January 1, 2017) (cited on page 61).
- [63] NIST. “Guide for Applying the Risk Management Framework to Federal Information Systems”. In: *NIST SP 800-37 revision 1* (February 2010). doi:10.6028/NIST.SP.800-37r1 (cited on pages 61, 149).
- [64] NIST. “Guide for Conducting Risk Assessments”. In: *NIST SP 800-30 revision 1* (September 2012). doi:10.6028/NIST.SP.800-30r1 (cited on pages 20, 21, 26, 27, 148, 149).
- [65] NIST. “Security and Privacy Controls for Federal Information Systems and Organizations”. In: *NIST SP 800-53 revision 4* (April 2013). doi:10.6028/NIST.SP.800-53r4 (cited on pages 62, 150).
- [66] NIST. “Assessing Security and Privacy Controls in Federal Information Systems and Organizations”. In: *NIST SP 800-53A revision 4* (December 2014). doi:10.6028/NIST.SP.800-53Ar4 (cited on pages 62, 69, 152).
- [67] NIST. *National Vulnerability Database*. 2016. <https://nvd.nist.gov/> (retrieved on October 11, 2016) (cited on page 1).
- [68] Lotfi ben Othmane et al. “Extending the Agile Development Process to Develop Acceptably Secure Software”. In: *IEEE Transactions on Dependable and Secure Computing* 11.6 (November 2014), pages 497–509. ISSN 1545-5971. doi:10.1109/TDSC.2014.2298011 (cited on pages 2, 3, 12).
- [69] OWASP. *Fuzzing*. <https://www.owasp.org/index.php/Fuzzing> (retrieved on January 1, 2017) (cited on page 35).
- [70] OWASP. *JBroFuzz*. <https://www.owasp.org/index.php/JBroFuzz> (retrieved on January 1, 2017) (cited on page 135).
- [71] OWASP. *Jenkins*. <https://jenkins.io/> (retrieved on January 1, 2017) (cited on page 76).
- [72] OWASP. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project (retrieved on August 7, 2016) (cited on page 2).
- [73] OWASP. *OWASP Benchmark Project*. <https://www.owasp.org/index.php/Benchmark> (retrieved on January 1, 2017) (cited on page 133).
- [74] OWASP. *OWASP ZAP*. https://www.owasp.org/index.php/Main_Page (retrieved on August 7, 2016) (cited on page 34).
- [75] OWASP. *Source Code Analysis Tools*. https://www.owasp.org/index.php/Source_Code_Analysis_Tools (retrieved on January 1, 2017) (cited on page 34).

- [76] OWASP. *CLASP Security Principles*. 2006. https://www.owasp.org/index.php/CLASP_Security_Principles (retrieved on January 1, 2017) (cited on page 125).
- [77] OWASP. *Category:OWASP Top Ten Project*. 2013. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (retrieved on October 11, 2016) (cited on pages 2, 18, 86).
- [78] OWASP. *OWASP Testing Guide v4*. September 17, 2014. <https://www.owasp.org/images/1/19/OTGv4.pdf> (retrieved on October 22, 2016) (cited on pages 33, 46, 47, 71).
- [79] OWASP. *Application Threat Modeling*. 2015. https://www.owasp.org/index.php/Application_Threat_Modeling (retrieved on October 11, 2016) (cited on pages 6, 21, 22, 24, 25).
- [80] OWASP. *OWASP Dependency Check*. December 21, 2016. https://www.owasp.org/index.php/OWASP_Dependency_Check (retrieved on January 1, 2017) (cited on page 76).
- [81] Chris Peake. “Red teaming: The art of ethical hacking”. In: *SANS Institute* (2003) (cited on pages 34, 139).
- [82] Christoph Pohl and Hans-Joachim Hof. “Secure Scrum: Development of Secure Software with Scrum”. In: *ArXiv e-prints arXiv:1507.02992* (2015-07). <http://arxiv.org/abs/1507.02992> (cited on pages 2, 10, 86, 88).
- [83] Winston W. Royce. “Managing the Development of Large Software Systems”. In: *Proceedings of IEEE WESCON*. (Los Angeles). Volume 26. August 1970, pages 1–9 (cited on page 1).
- [84] John Rushby. “Security requirements specifications: How and what”. In: *Symposium on Requirements Engineering for Information Security (SREIS)*. Volume 441. Citeseer. 2001 (cited on pages 2, 43).
- [85] Paul Saitta, Brenda Larcom, and Michael Eddington. *Trike v.1 Methodology Document [Draft]*. July 2005. http://octotrike.org/papers/Trike_v1_Methodology_Document-draft.pdf (retrieved on October 22, 2016) (cited on page 42).
- [86] C. Schmittner, Z. Ma, and E. Schoitsch. “Combined safety and security development lifecycle”. In: *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. July 2015, pages 1408–1415. doi:10.1109/INDIN.2015.7281940 (cited on page 1).
- [87] Ken Schwaber. *Agile Project Management with Scrum*. 1st edition. Microsoft Press, 2004. ISBN 073561993x (cited on pages 4, 10, 17, 43, 46, 70).
- [88] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. July 2016. <http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf#zoom=100> (retrieved on August 7, 2016) (cited on pages 15, 16, 18, 45, 70, 86, 91).
- [89] *Scrum Release Planning*. http://www.scrum-institute.org/Release_Planning.php (retrieved on August 7, 2016) (cited on page 18).
- [90] Adam Shostack. *Threat Modeling: Designing for Security*. 1st edition. John Wiley & Sons Inc, April 25, 2014. ISBN 1118809998 (cited on page 21).
- [91] Jochen Peter Sondermann. “Interne Qualitätsanforderungen und Anforderungsbewertung”. In: *Masing, & Pfeifer (Hrsg.), Handbuch Qualitätsmanagement 5* (2007), pages 387–404 (cited on page 88).
- [92] Kevin Stine et al. “Volume 1: Guide for Mapping Types of Information and Information Systems to Security Categories”. In: *NIST SP 800-60 Volume 1 revision 1* (August 2008). doi:10.6028/NIST.SP.800-60v1r1 (cited on pages 60, 76, 77, 146).

- [93] Marianne Swanson et al. “Contingency Planning Guide for Federal Information Systems”. In: *NIST SP 800-34 revision 1* (May 2010). doi:10.6028/NIST.SP.800-34r1 (cited on pages 61, 147).
- [94] Frank Swiderski and Window Snyder. *Threat Modeling*. 1st edition. Microsoft Press, 2004. ISBN 0735619913 (cited on page 22).
- [95] Synopsis. *Static Code Analysis: Coverity*. <https://www.synopsys.com/software-integrity/products/static-code-analysis.html> (retrieved on January 1, 2017) (cited on page 133).
- [96] Siiskonen Tuuli, Särs Camillo, and Vähä-Sipilä Antti. “Generic Security User Stories”. In: *Handbook of the Secure Agile Software Development Life Cycle*. Edited by Pietikäinen Pekka and Röning Juha. Oulu: University of Oulu, 2014. Chapter 2, pages 9–14 (cited on pages 2, 10).
- [97] Antti Vähä-Sipilä. *Product Security Risk Management in Agile Product Management*. 2010. https://owasp.org/images/c/c6/OWASP_AppSec_Research_2010_Agile_Prod_Sec_Mgmt_by_Vaha-Sipila.pdf (retrieved on August 7, 2016) (cited on pages 2, 3, 11–13).
- [98] *Valgrind*. <http://valgrind.org/> (retrieved on August 7, 2016) (cited on page 34).
- [99] Ylimannela Ville and Helenius Marko. “Security Activities in Scrum Control Points”. In: *Handbook of the Secure Agile Software Development Life Cycle*. Edited by Pietikäinen Pekka and Röning Juha. Oulu: University of Oulu, 2014. Chapter 4, pages 22–28 (cited on pages 2, 12).
- [100] Jaana Wäyrynen, Marine Bodén, and Gustav Boström. “Security Engineering and eXtreme Programming: An Impossible Marriage?” In: *Extreme Programming and Agile Methods - XP/Agile Universe 2004: 4th Conference on Extreme Programming and Agile Methods*. (Calgary, Canada). Edited by Carmen Zannier, Hakan Erdogmus, and Lowell Lindstrom. Berlin, Heidelberg: Springer Berlin Heidelberg, August 15, 2004, pages 117–128. ISBN 9783540277774. doi:10.1007/978-3-540-27777-4_12. http://dx.doi.org/10.1007/978-3-540-27777-4_12 (cited on pages 2, 9, 10, 43).
- [101] Dave West et al. “Agile development: Mainstream adoption has changed agility”. In: *Forrester Research 2.1* (January 2010), page 41. http://programmedevelopment.com/public/uploads/files/forrester_agile_development_mainstream_adoption_has_changed_agility.pdf. (cited on page 1).
- [102] Zeki Yazar. *A qualitative risk analysis and management tool—CRAMM*. 2002. <https://www.sans.org/reading-room/whitepapers/auditing/qualitative-risk-analysis-management-tool-cramm-83> (retrieved on January 1, 2017) (cited on page 59).