

Integration von Cyber Security innerhalb agiler Prozessmodelle am Beispiel von Scrumban

Andreas Pilgerstorfer



BACHELORARBEIT

eingereicht am
Fachhochschul-Bachelorstudiengang
Kommunikation, Wissen, Medien
in Hagenberg

im Juni 2022

Betreuung:

FH-Prof. DI (FH) Dr. Johannes Schönböck

© Copyright 2022 Andreas Pilgerstorfer

Alle Rechte vorbehalten

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt. Die vorliegende, gedruckte Arbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Hagenberg, am 20. Juni 2022

Andreas Pilgerstorfer
Andreas Pilgerstorfer (Jun 18, 2022 16:06 GMT+2)

Andreas Pilgerstorfer

Inhaltsverzeichnis

Erklärung	iv
Kurzfassung	vii
Abstract	viii
1 Einleitung	1
1.1 Zielsetzung der Arbeit	2
1.2 Aufbau der Arbeit	2
2 Theoretische Grundlagen zu agilen Prozessmodellen	3
2.1 Grundlagen agiler Prozesse	3
2.2 Scrum	5
2.2.1 Scrum Team	5
2.2.2 Scrum Events	6
2.2.3 Scrum Artefakte	7
2.3 Kanban	7
2.4 Scrumban	9
2.5 Probleme von etablierten agilen Prozessmodellen in Bezug auf Cyber Security	11
3 Grundlagen bestehender Modelle zur Integration von Security in den Soft- wareentwicklungsprozess	14
3.1 Methodische Vorgehensweise bei der Literaturanalyse	15
3.2 Bestehende Secure Software Development Modelle	16
3.3 Erkenntnisse aus bestehenden Modellen	17
4 Integration von Security Maßnahmen in den Scrumban Prozess	20
4.1 Der menschliche Faktor	20
4.1.1 Security Training	20
4.1.2 Security Rollen	21
4.2 Security durch den Backlog	22
4.2.1 Security User Stories	22
4.2.2 Security Anforderungen	24
4.2.3 Evil User Stories	28
4.2.4 Klassifizierung von Security User Stories	30

4.2.5	Security Akzeptanzkriterien	32
4.3	Tools und Praktiken zur Einhaltung von Security Maßnahmen während Entwicklung, Testung und Deployments	33
4.3.1	Software Composition Analysis	35
4.3.2	Static Application Security Testing	37
4.3.3	Dynamic Application Security Testing	38
4.3.4	Security Unit Testing	39
4.4	Secure Scrumban	40
4.4.1	Secure Code Review	40
4.4.2	Security Retrospektive	41
4.4.3	Secure Scrumban Prozess	41
5	Fazit	43
	Abbildungsverzeichnis	45
	Quellenverzeichnis	47

Kurzfassung

Die Berücksichtigung von nicht-funktionalen Anforderungen, wie Cyber Security, stellt in der modernen Softwareentwicklung eine große Herausforderung dar, der man sich stellen sollte, da die Anzahl an Cyberangriffen laufend ansteigt und die Angriffsvarianten sich ständig weiterentwickeln. Vor allem im Bereich der agilen Softwareentwicklung ist die Einbindung von Security Maßnahmen besonders schwierig, da sich die Anforderungen der Projekte kontinuierlich ändern. Im Rahmen dieser Arbeit wird auf bereits bestehende Modelle zur Integration von Cyber Security in den Softwareentwicklungsprozess eingegangen und deren Praktiken evaluiert. In weiterer Folge werden die gefundenen Best Practices in den Scrumban Prozess inkludiert.

Bei der Einführung von Maßnahmen gilt es zu beachten, dass diese so früh wie möglich in den Prozess eingeführt werden, um die mit der Entdeckung einer Schwachstelle verbundenen Kosten so gering wie möglich zu halten. Somit beginnen die Maßnahmen bereits bei der Planung und Sammlung der Anforderungen, welche im Product Backlog festgehalten werden. Zur Erhebung von Security Anforderungen können Security Design Prinzipien, Threat Modeling aber auch Evil User Stories eingesetzt werden. Die erhobenen Anforderungen können anschließend in Form von Security User Stories nach ihrem Schweregrad priorisiert in den Backlog eingeordnet werden. Für die Priorisierung kann der CVSS-Score berechnet werden, um den Schweregrad des Security Problems zu ermitteln. Security Probleme mit hohem Schweregrad sollten dabei mit erhöhter Priorität umgesetzt werden. Um die Umsetzung der Arbeitsitems sicherer zu gestalten, können automatisierte Tools eingesetzt werden, die auf die Einhaltung von sicherer Softwareentwicklung achten. In der Arbeit wird ebenfalls die Einführung von neuen Security Rollen motiviert, die dazu dienen, Security Entscheidungen zu treffen sowie als Kontaktpersonen für Fragen zur Verfügung zu stehen. Zusätzlich wird die Einführung von zwei neuen Events in den Scrumban Prozess vorgeschlagen. Abschließend wird im Zuge dieser Arbeit ein Secure Scrumban Prozess vorgestellt, der die zuvor beschriebenen Praktiken und Tools berücksichtigt.

Abstract

The consideration of non-functional requirements, such as cyber security, represents a major challenge in modern software development. Due to the fact that the number of cyber attacks are steadily increasing and attack variants are constantly evolving it is getting more and more important to tackle these threats. Especially in the area of agile software development, the integration of security measures is particularly difficult, since the requirements of the projects are constantly changing. This paper evaluates already existing models for the integration of cyber security in the software development process and analyses their practices as well. Subsequently, these results will be included in the Scrumban process.

When carrying out measures, it is important to ensure that all measures are included in the process as soon as possible. This is necessary to keep the costs, which are associated with discovering a vulnerability, as low as possible. Therefore, some measures are already being introduced during the planning and collection of requirements. In order to collect the security requirements, it is possible to use security design principles, threat modeling and evil user stories. These requirements can be used in order to create security user stories which should be included prioritized according to their severity within the product backlog. In order to perform the prioritization, the CVSS score of the respective vulnerabilities can be calculated. The CVSS score shows the severity of the vulnerability. Issues with a high severity should be implemented with a higher priority. To ensure secure coding practices automated security testing tools can be used during development and deployment of the software. This thesis will also motivate the introduction of new security roles. These roles are needed in order to make security related decisions as well as being contact persons for any security questions which might occur during the development process. Additionally this thesis proposes the introduction of two new events within the Scrumban process in order to make it more secure. Finally, a new Secure Scrumban process is presented which includes all of the earlier described security practices and tools.

Kapitel 1

Einleitung

Durch die wachsende Digitalisierung in unterschiedlichen Bereichen wächst auch das Risiko für Privatpersonen und Unternehmen, Opfer von digitaler Kriminalität zu werden. Aufgrund der globalen Covid-19-Pandemie mussten viele Unternehmen ihre Distributionswege in die digitale Welt verlagern, wodurch das Risiko stark angestiegen ist. Alleine innerhalb der ersten Hälfte des Jahres 2021 stieg die Anzahl an weltweiten Cyberattacken um 125 % im Vergleich zum Vorjahr [49]. Pro Unternehmen wurden im Jahr 2021 durchschnittlich 270 Attacken verzeichnet. Dies stellt im Vergleich zum Jahr 2020 einen Anstieg von 31 % dar [6]. Der am stärksten von Cyberangriffen betroffene Wirtschaftssektor ist der Konsum- und Dienstleistungssektor [49].

Das Ausmaß der Attacken lässt sich weiters durch die steigenden Kosten für Unternehmen durch Cyberangriffe verdeutlichen. Schätzungen zu Folge werden Cyberangriffe ab 2025 jährlich weltweite Kosten von 10,5 Trillionen US-Dollar verursachen [68]. Es sind jedoch nicht nur große Firmen von solchen Angriffen betroffen. 43 % aller Cyberattacken im Jahr 2019 richteten sich gegen kleine Unternehmen. Von diesen waren nur gerade einmal 14 % auf Angriffe vorbereitet [5]. Die Folgen einer Attacke können jedoch große Auswirkungen auf den Umsatz, das Vertrauen der Kund:innen, die Online-Verfügbarkeit und auf die Reputation des Unternehmens haben [68].

Deshalb stellt die Entwicklung von sicheren Anwendungen einen essentiellen Bestandteil für den nachhaltigen Erfolg von Unternehmen dar. Traditionellerweise fokussieren Softwareentwicklungsprozesse, wie der Software Development Lifecycle, allerdings primär nur auf die fristgerechte Umsetzung von neuen Funktionalitäten. Der Security Faktor wird dabei oftmals nicht berücksichtigt [29]. Auch agile Methoden wie Extreme Programming, Feature Driven Development, Kanban, Scrum oder Mischformen wie Scrumban werden oftmals dafür kritisiert, keine Sicherheitspraktiken in ihre Prozesse zu inkludieren [2]. Aus diesem Grund stellt die Einführung eines agilen Secure Software Development Lifecycles einen wichtigen Schritt zur Optimierung des Entwicklungsprozesses innerhalb von Unternehmen dar.

1.1 Zielsetzung der Arbeit

Das Ziel dieser Arbeit ist es, die Probleme und Schwachstellen von agilen Prozessmodellen in Bezug auf Cyber Security aufzuzeigen sowie Verbesserungsmaßnahmen in Form von Tools und Best Practices am Beispiel von Scrumban zu diskutieren und innerhalb eines adaptierten Prozesses darzustellen. Dabei werden auf Basis der durchgeführten Recherche ausgewählte Praktiken und Tools näher betrachtet und mittels einer exemplarischen praktischen Umsetzung anhand von Softwareprojekten und Prozessen des Unternehmens Tractive¹ realisiert.

Folgende Fragestellungen sollen im Zuge der Arbeit beantwortet werden:

- Welche Secure Software Development Modelle existieren bereits?
- Durch welche Best Practices, Technologien, Tools und Methoden können die Phasen agiler Prozesse am Beispiel von Scrumban sicherer gestaltet werden?
- Wie könnte ein agiler Scrumban Prozess unter Berücksichtigung von Cyber Security Maßnahmen aussehen?

1.2 Aufbau der Arbeit

Agile Prozessmodelle stellen eine Möglichkeit dar, um mittels iterativer Verbesserung Produkte unabhängig von sich ändernden Anforderungen zu entwickeln. Kapitel 2 geht deshalb auf die theoretischen Grundlagen agiler Prozessmodelle ein. Dabei werden drei ausgewählte Modelle (Scrum, Kanban, Scrumban) näher betrachtet. Anschließend wird der Begriff der Cyber Security definiert und aufgezeigt, welche Probleme und Herausforderungen in der Integration von Sicherheitsmaßnahmen in agilen Prozessen existieren.

Kapitel 3 beschäftigt sich mit bestehenden Secure Software Development Lifecycle Modellen. Es wird erklärt, was unter solchen Modellen verstanden wird und ein Überblick über aktuell existierende Modell gegeben. Dabei wird die Quintessenz der unterschiedlichen Modelle hervorgehoben und auf die identifizierten Gemeinsamkeiten und Unterschiede eingegangen.

In Kapitel 4 werden die auf Basis der Recherche gefundenen Tools und Best Practices vorgestellt und deren Einsatz in einem Scrumban Prozess und die Sinnhaftigkeit eines Einsatzes dargelegt.

Ein abschließendes Fazit zu den Ergebnissen der Recherche und eine Diskussion zur Einführung von security-spezifischen Praktiken in Kapitel 5 rundet die vorliegende Arbeit ab.

¹<https://tractive.com/de/>

Kapitel 2

Theoretische Grundlagen zu agilen Prozessmodellen

Agile Prozessmodelle sind aus der modernen Softwareentwicklung kaum mehr wegzudenken. Dieses Kapitel gibt einen Überblick über die Grundwerte von agilen Vorgehensweisen. Außerdem werden die Kernkonzepte konkreter agiler Prozessmodelle vorgestellt. Hierbei wird auf Scrum, Kanban und Scrumban eingegangen. Anschließend wird der Begriff Cyber Security definiert und auf die Probleme agiler Prozesse bei der Integration von Security Maßnahmen eingegangen.

2.1 Grundlagen agiler Prozesse

Anfang der 1990er-Jahre begann ein Umdenken im Bereich der Prozessmodelle zur Entwicklung von Softwarelösungen [9]. Aufgrund der immer schneller fortschreitenden Technologiezyklen und den daraus resultierenden ständigen Änderungen von Anforderungen, stellt die vollständige Sammlung aller Erfordernisse eines Projekts in der ersten Projektphase aufgrund der fehlenden Flexibilität bestehender Prozessmodelle eine schwer lösbare Aufgabe dar. Um diesen Herausforderungen entgegenzuwirken, wurden verschiedene Modelle entwickelt, die mehrheitlich auf dem Konzept des iterativ-inkrementellen Vorgehen von Basili und Turner basieren [4]. Hierbei wird zu Beginn eine erste Schätzung der gesamten finalen Anforderungen des Endproduktes gemacht und eine initiale Umsetzung durchgeführt. Die gesamten Anforderungen eines Projekts werden dabei in einer sogenannten Projektkontrollliste festgehalten. Anschließend wird die ursprüngliche Umsetzung in Zyklen iterativ um weitere Anforderungen aus der Projektkontrollliste ergänzt und verbessert, wie Abbildung 2.1 zeigt. Hierfür werden in jedem Schritt Einträge aus der Liste ausgewählt und umgesetzt. Anschließend wird die erfolgte Teilumsetzung analysiert und die Projektkontrollliste aktualisiert. Diese Iterationen werden so lange fortgeführt bis die Liste keine Einträge mehr enthält und eine finale Lösung erreicht wurde. Der Vorteil dieser Vorgehensweise liegt darin, dass es nicht notwendig ist zu Beginn alle Anforderungen vollständig zu definieren. Es müssen nur die Anforderungen für die aktuelle Iteration definiert werden. Auf Basis der iterativ-inkrementellen Vorgehensweise wurde eine Vielzahl an unterschiedlichen Ansätzen für den flexiblen Umgang mit sich ändernden Anforderungen entwickelt [9].

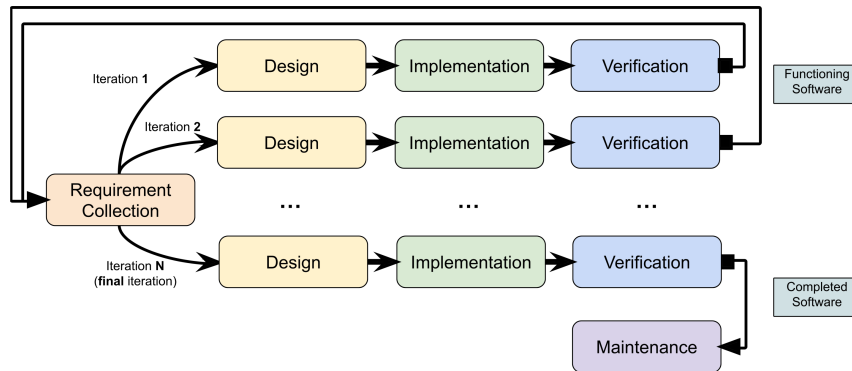


Abbildung 2.1: Schritte des iterativ-inkrementellen Vorgehens [45]

Um die verschiedenen Ansätze zu vereinheitlichen, wurde 2001 ein Kongress bestehend aus 17 führenden Vertreter:innen unterschiedlicher agiler Prozessmodelle, wie zum Beispiel Scrum, Feature-Driven Development oder Extreme Programming (XP) abgehalten, um gemeinsame einheitliche Charakteristiken von agilen Prozessen zu definieren. Als Ergebnis wurden zwölf Grundprinzipien von agilen Prozessen erarbeitet. Diese Prinzipien werden als das agile Manifest bezeichnet [55]. Die oberste Priorität hat dabei, den jeweiligen Kund:innen frühestmöglich und durchgehend neue Softwarelösungen zu liefern. Dabei spielt es keine Rolle, ob sich die Anforderungen der Kund:innen während des Entwicklungsprozesses ändern, da die ständige Veränderung als positiver Einfluss auf das Endprodukt gesehen wird. Die kontinuierlichen Änderungen sind möglich, da innerhalb kurzer und über die Dauer des Prozesses gleichmäßiger Zeitabstände Zwischenprodukte geliefert werden. Um den Fortschritt innerhalb der Agilität zu gewährleisten, muss der Fokus ständig auf technisch funktionierende Software mit adäquatem Design gelegt werden. Dabei stellt auch die Einfachheit der Prozesse einen wichtigen Bestandteil dar. Dies bezieht sich auf die Menge an nicht durchgeführter Arbeit, welche keinen speziellen Mehrwert für die Erstellung der Leistungen für die Kund:innen haben, wie beispielsweise die übermäßige Dokumentation der Projekte. Neben der kontinuierlichen Auslieferung von Softwarelösungen an die Kund:innen sind die Individuen innerhalb der Projektteams ein Kernelement agiler Prozesse. Hierbei soll vor allem auf selbstorganisierte Teams gesetzt werden, die aus motivierten Personen bestehen. Innerhalb eines Teams stellt die Zusammenarbeit von Expert:innen und Entwickler:innen einen wichtigen Aspekt dar. Um die Zusammenarbeit zu fördern, stellen persönliche Gespräche innerhalb agiler Prozesse die effizienteste Methode dar, um Informationen innerhalb des Teams zu kommunizieren. Zur Verbesserung der team-internen Effektivität sollten in regelmäßigen Zeitabständen die Prozesse reflektiert und nach Bedarf erweitert oder angepasst werden [51]. Neben den zwölf Grundprinzipien umfasst das Manifest auch vier Werte der agilen Softwareentwicklung:

1. „*Individuals and interactions over processes and tools*“ [51].
Der Austausch mit den Kund:innen stellt eine wichtigere Komponente für das Projektergebnis dar als die eingesetzten Prozesse und Werkzeuge [22].

2. „*Working software over comprehensive documentation*“ [51].
Hierdurch wird ausgedrückt, dass die Qualität der Software eine hohe Bedeutung hat, da nur durch funktionierende Produkte langfristige Beziehungen zu den Kund:innen aufgebaut werden können. Der Dokumentation wird dabei eine niedrigere Bedeutung zugewiesen [22].
3. „*Customer collaboration over contract negotiation*“ [51].
Hiermit wird erneut betont, dass der Austausch mit den Kund:innen wichtiger ist als das Aufsetzen von formellen juristischen Verträgen [22].
4. „*Responding to change over following a plan*“ [51].
Durch agile Softwareentwicklung soll gewährleistet werden, dass schnell und flexibel auf sich ändernde Anforderungen reagiert werden kann [22].

Abschließend lässt sich agile Softwareentwicklung als ein Oberbegriff für eine Sammlung von Modellen, Frameworks und Praktiken definieren, die auf den Werten und Prinzipien des agilen Manifests basieren [50]. Auf die drei meistgenutzten Modelle, Scrum, Kanban und Scrumban, basierend auf einer Erhebung im Zuge des „15th Annual State Of Agile Report“ [15], wird in den Abschnitten 2.2, 2.3 und 2.4 eingegangen.

2.2 Scrum

Scrum ist ein agiles Prozessmodell, welches in den frühen 1990er-Jahren von Jeff Sutherland und Ken Schwaber entwickelt wurde [41]. Scrum gilt als das meistgenutzte agile Modell und basiert auf einem iterativ-inkrementellen Ansatz, um die Vorhersagbarkeit und die Risikokontrolle eines Prozesses zu optimieren. Das Modell baut auf vier formalen Events (Sprint Planning Meeting, Daily Scrum Meeting, Sprint Review und Sprint Retrospektive, siehe Abschnitt 2.2.2) auf, um eine aktuelle Bearbeitungsperiode, einen sogenannten Sprint, bestmöglich zu überblicken und gegebenenfalls anzupassen. Neben den Events besteht das Scrum Framework aus dem Scrum Team, siehe Abschnitt 2.2.1, und den Scrum Artefakten, siehe Abschnitt 2.2.3 [41].

2.2.1 Scrum Team

Ein Scrum Team besteht aus einer kleinen Anzahl von Personen, idealerweise weniger als zehn, und sonst keinen weiteren Subeinheiten. Schwaber und Sutherland argumentieren, dass größere Teams weniger produktiv sind und die Kommunikation im Team schwieriger wird. Das Team ist eine Einheit, die sich nur auf eine Zielsetzung, das Produktziel, fokussieren muss. Die Teams sind cross-funktional besetzt, das heißt die Teammitglieder:innen besitzen bereits alle nötigen Fähigkeiten zur Erreichung des Produktziels. Die Mitglieder:innen sind außerdem angehalten, selbstständig intern darüber zu entscheiden, wer, was, wann und auf welche Art und Weise erledigt. Jedes Scrum Team besteht aus drei unterschiedlichen Rollen, den Entwickler:innen, dem oder der Scrum Master:in und dem oder der Product Owner:in. Die Entwickler:innen sind dafür verantwortlich, in jedem Sprint eine benutzbare Weiterentwicklung des Produktes zu entwickeln.

Der oder die Product Owner:in ist dafür verantwortlich, das bestmögliche Resultat am Ende eines Sprints zu erreichen. Diese Rolle beschäftigt sich primär mit der Entwicklung und Kommunikation des Produktziels sowie mit der Erstellung, Kommunikation

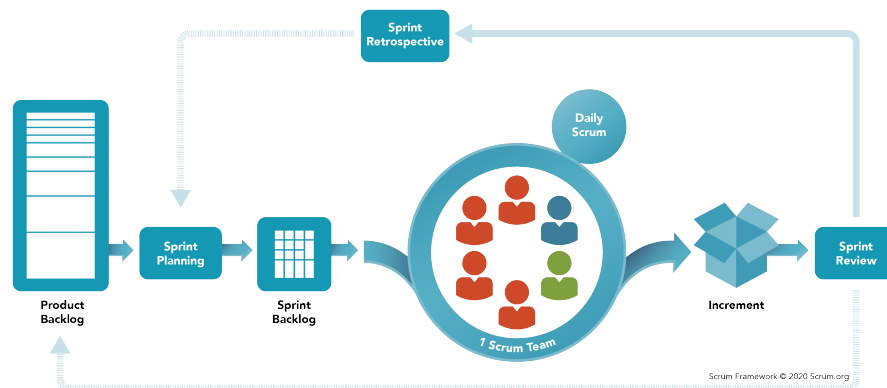


Abbildung 2.2: Scrum Prozess [48]

und Priorisierung des Product Backlogs, siehe Abschnitt 2.2.3. Die Rolle wird von nur einer Person ausgeführt, die die Bedürfnisse aller Stakeholder:innen durch den Product Backlog repräsentiert. Der oder die Product Owner:in ist auch die einzige Person, die den Backlog verändern darf.

Die dritte Rolle eines Scrum Teams ist die des oder der Scrum Master:in. Diese Person ist dafür verantwortlich, dass alle Teammitglieder:innen den Scrum Prozess verstehen und bei Bedarf Hilfestellungen zur Einhaltung der Regeln zu leisten. Dabei ist das oberste Ziel, die Effektivität des Teams zu erhöhen und die Ausübung der Scrum Praktiken zu verbessern. Zusätzlich ist der oder die Scrum Master:in dafür verantwortlich, den Scrum Prozess mit allen Events in einer produktiven Weise abzuhalten und alle möglichen Hindernisse im Scrum Prozess zu entfernen. Außerdem werden Scrum Master:innen für die Einführung von Scrum Prozessen in Organisationen eingesetzt [41].

2.2.2 Scrum Events

Die zentrale Einheit eines Scrum Prozesses ist der Sprint. Ein Sprint stellt eine Zeiteinheit dar, in der ausgewählte Backlog Items und alle weiteren Scrum Events abgehalten werden, siehe Abbildung 2.2. Ein Sprint hat eine fixierte Länge, die maximal einen Monat dauert. Durch die kurze Sprintlänge kann agil auf Veränderungen der Anforderungen eingegangen werden. Wenn ein Sprint abgeschlossen wird, beginnt sofort der nächste. Die Länge des Sprints bleibt dabei jedoch immer gleich, was die Vorhersagbarkeit weiterer Sprints erleichtert. Die Events eines Sprints stellen Abstimmungen innerhalb des Teams dar und setzen alle auf das Konzept des Timeboxings, wodurch jedes Meeting eine vordefinierte maximale Dauer zugewiesen bekommt.

Zu Beginn jedes Sprints wird das Sprint Planning Meeting abgehalten. Hierbei plant das gesamte Scrum Team den anstehenden Sprint. Zu Beginn des Meetings wird das Ziel des kommenden Sprints definiert. Anschließend wird gemeinsam entschieden, welche Items aus dem Product Backlog im Sprint durchgeführt werden sollten [41]. Dafür können Methoden wie das Scrum Planning Poker verwendet werden. Hierbei stimmen die Teilnehmer:innen mit Punkten ab, wie aufwändig ein Item aus dem Backlog ist [3].

Jeden Tag findet das Daily Scrum Meeting statt. In diesem maximal 15-minütigen Meeting wird der aktuelle Fortschritt in Anbetracht des Sprintziels evaluiert. Dabei wird auf Probleme eingegangen, die für den aktuellen Arbeitstag von Bedeutung sind.

Am Ende eines Sprints findet das Sprint Review statt. Hierbei wird das Ergebnis des Sprints den wichtigsten Stakeholder:innen vorgestellt und gemeinsam über mögliche Anpassungen des Projektziels diskutiert. Dies führt zu potentiellen Anpassungen des Product Backlogs. Das Meeting sollte eine Länge von vier Stunden nicht überschreiten.

Neben dem Sprint Review findet am Ende des Sprints auch die Sprint Retrospektive statt. Hierbei diskutiert das Scrum Team intern über Möglichkeiten wie sich die Qualität und die Effektivität innerhalb des Sprints verbessern lässt. Probleme und positive Aspekte des letzten Sprints werden besprochen und Lösungen für die identifizierten Probleme diskutiert, die im nächsten Sprint umgesetzt werden sollen. Die Sprint Retrospektive ist für eine Länge von drei Stunden angesetzt [41].

2.2.3 Scrum Artefakte

Eines der Artefakte von Scrum ist der Product Backlog. Dieser stellt eine geordnete Liste dar, die alles enthält, was benötigt wird, um das Produkt zu verbessern beziehungsweise das Projektziel zu erreichen. Die Backlog Items können vom gesamten Scrum Team erstellt werden. Große Backlog Items müssen noch weiter zu kleineren Einheiten von den Entwickler:innen heruntergebrochen werden.

Die Items des Product Backlogs, die innerhalb eines Sprints erfüllt werden sollen, werden im Rahmen des Sprint Planning Meetings in den Sprint Backlog, ein weiteres Artefakt, übernommen. Diese tragen dazu bei, das Sprintziel zu erfüllen.

Ein weiteres Artefakt ist der Inkrement. Ein Inkrement stellt einen Meilenstein oder ein Teilergebnis in Bezug auf das Projektziel dar und baut auf die vorherigen Inkremente auf. Jedes Item im Backlog hat eine sogenannte Definition of Done. Diese gibt an unter welchen Bedingungen das Item als erledigt gilt. Jedes Mal, wenn die Definition of Done erfüllt wird, entsteht ein neuer Inkrement. Am Ende eines Sprints wird die Summe aller Inkremente den Stakeholder:innen präsentiert [41].

2.3 Kanban

Kanban ist eine agile Prozessmethode, die auf das Management, die Verbesserung und die Definition von Wissensarbeiten abzielt. Darunter fallen Arbeiten für physische Produkte und Dienstleistungen aber auch Softwareprodukte [1]. Die Methode wurde in den frühen 1940er-Jahren von Taiichi Oho für den japanischen Automobilhersteller Toyota entwickelt, um die Produktivität der Arbeitsschritte zu steigern [54]. Im Vergleich zu Scrum, siehe Abschnitt 2.2, gibt Kanban keine spezifischen neuen Rollen und Positionen vor. Die Kanban Methode besteht aus vier Prinzipien und sechs Praktiken [1].

Das erste Prinzip von Kanban lautet, starte mit dem was du gerade tust. Dabei ist es jedoch laut dem zweiten Prinzip relevant, die aktuell gültigen Verantwortlichkeiten, Prozesse und Rollen im Unternehmen zu respektieren. Das dritte Prinzip gibt vor, dass die Veränderungen durch Kanban schrittweise und nicht auf einmal eingeführt werden sollten. Das vierte Kanban Prinzip besagt, dass auf allen Ebenen in der Organisation Führung und Verantwortung für die Tätigkeiten übernommen werden soll [33].

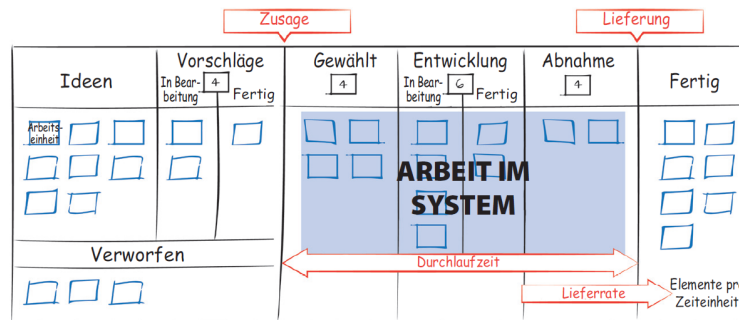


Abbildung 2.3: Beispiel eines Kanban Boards [1]

Kanban fokussiert auf die Visualisierung der Prozesse. Hierfür wird ein Kanban Board, siehe Abbildung 2.3, eingesetzt. Dieses dient dazu, die Abläufe des Arbeitsprozesses übersichtlich darzustellen. Die Spalten des Boards stellen die verschiedenen Schritte dar, die im Zuge der Arbeit durchlaufen werden. Um eine feinere Unterscheidung innerhalb einer Spalte vorzunehmen, können Swimlanes verwendet werden. Diese werden auf dem Board durch horizontale Linien eingezeichnet. Swimlanes können für die Einteilung nach Prioritäten innerhalb der Spalte eingesetzt werden. Zusätzlich sollen auch die Regeln des Prozesses visualisiert werden. Hierfür kann über den Spalten festgehalten werden, was erreicht werden muss, damit ein Element von einer Spalte in die nächste Spalte bewegt werden darf. Das Kanban Board visualisiert ebenfalls die Punkte für die Zusage (Commitment) und die Lieferung der Arbeit an die Kund:innen. Die Gestaltung des Boards wird nicht fest vorgegeben und ist je nach Einsatzgebiet individuell anpassbar. Vor dem Commitment kann aus unterschiedlichen Items oder Ideen ausgewählt werden. Danach startet der Prozess bis die Lieferung erreicht wird, da hier die Arbeit an dem Item als abgeschlossen gilt. Die Zeit, die zwischen Commitment und Lieferung vergeht, wird als Durchlaufzeit bezeichnet und als Metrik zur Messung des Kanbanprozesses herangezogen [1].

Eine weitere Praktik stellt die Einführung von Work in Progress (WIP) Limits dar. Diese stellen sicher, dass Kanban keinem Push-Ansatz¹ sondern dem Pull-Prinzip folgt. Hierbei dürfen keine neuen Arbeiten gestartet werden, solange vorher gestartete Arbeiten noch nicht abgeschlossen wurden. Dies verhindert, dass viele unfertige Arbeiten gerade in Bearbeitung sind, wodurch die Durchlaufzeiten der Items erhöht werden. Die WIP-Limits legen pro Spalte auf dem Kanban Board fest, wie viele Items dort maximal platziert werden dürfen. Ist das WIP-Limit erreicht, müssen somit zuerst Items aus dieser Spalte weiterentwickelt werden, um diese verschieben zu können. Das Verschieben der Einheiten auf dem Board visualisiert das Fluss-Paradigma von Kanban, bei dem die Arbeit als ein Fluss zur Erzeugung eines Mehrwerts dient [1].

Durch Rückkopplungsschleifen werden in Kanban in zyklischen Abständen Meetings abgehalten, um die Erbringung der Leistung weiterzuentwickeln. Der Zeitabstand zwischen den Meetings wird als Kadenz bezeichnet. Kanban setzt auf sieben Kadenzen aus den Bereichen Lieferplanung, Arbeitsfluss, Zusagepunkte, Service, Risiko, Operatives

¹Einheiten werden nach festgelegten Zeiten oder Mengen zur Entwicklung gegeben [1].

Geschäft und Strategie. Die Dauer einer Kadenz ist dabei je nach Verwendungskontext des Prozessmodells variabel anzupassen. Außerdem können mehrere Bereiche auch in nur einer Kadenz behandelt werden [1].

2.4 Scrumban

Ken Schwaber, Mitbegründer von Scrum, geht davon aus, dass 75 % aller Organisationen, die Scrum verwenden, nicht die Vorteile erreichen werden, die sie sich von dem Prozessmodell erhoffen. Trotz der Simplizität des Modells ist es nicht einfach, Scrum in einer Organisation erfolgreich umzusetzen, da sowohl Individuen als auch Teams Hindernisse im Umgang mit den Strukturen und Prozessen von Scrum überwinden müssen. Die Events von Scrum nehmen eine wichtige Rolle innerhalb des Modells ein. Um die Hürde der Einführung zu minimieren, verzichten viele Organisationen auf bestimmte Events oder modifizieren deren Bestimmung. Dies führt jedoch nur selten zu dem gewünschten Ergebnis [33]. Corey Ladas führte 2008 in seinem Buch den Begriff Scrumban ein. Ladas beschreibt damit ursprünglich ein Modell, das Softwareentwicklungsteams von Scrum zu einem anderen Framework wie Kanban überführt [26]. Reddy beschreibt zusätzlich noch weitere Einsatzmöglichkeiten von Scrumban. Das Modell hilft Organisationen und Teams, Herausforderungen bei der Einführung von Scrum zu bewältigen. Zusätzlich hilft Scrumban Scrum-ähnliche Prozesse und Praktiken in einem Unternehmen effektiv einzuführen [33]. Der Scrumban Prozess stellt eine Kombination der beiden Prozessmodelle Scrum und Kanban dar. Scrumban basiert auf Kanban Prinzipien, wie dem Pull Prinzip, den WIP-Limits und der Visualisierung der Einheiten oder User Story Cards auf dem Kanban Board. Von Scrum werden Events, wie das Review, die Retrospektive sowie das Daily Standup Meeting aufgegriffen.

Einer der Hauptprinzipien von Scrumban ist der Verzicht auf das von Scrum bekannte Prinzip der Sprints, des Timeboxings und der Planung des Sprint Backlogs. Anstelle von Sprints setzt Scrumban auf einen kontinuierlichen Arbeitsfluss. Ladas kritisiert dabei an Scrum, dass durch die Vorausplanung für eine Sprintlänge im Rahmen des Sprint Planning Meetings sich mehr Items als notwendig im Backlog eines Sprints befinden. Er empfindet jegliche Einheiten im Backlog, die darüber hinausgehen, dass das Entwicklungsteam ohne Verzögerung eine neue Arbeit beginnen kann, als überflüssig, da dadurch nur ein großer Sprint Backlog entsteht [26].

Das Scrumban Board, siehe Abbildung 2.4, sollte neben der Abbildung des für jedes Team individuellen Entwicklungsprozesses durch Spalten, auch noch weitere Spalten enthalten. Alle User Stories, die sich im Product Backlog befinden, werden in einer Spalte zu Beginn des Boards visualisiert. Zusätzlich wird eine sogenannte Ready-Spalte eingeführt. In diese Spalte werden alle Items aus dem Backlog gezogen, die bereit zur Umsetzung sind. Die Priorität der Items wird dabei über die Reihenfolge innerhalb der Spalte dargestellt [26]. Zusätzlich können, wie bei Kanban, Swimlanes verwendet werden, um die Priorität noch besser auf dem Board darzustellen [33]. Zu dem Zeitpunkt wo die Einheiten in der Ready-Spalte sind, sind jedoch noch keine Entwickler:innen den jeweiligen Items zugeordnet. Manche Scrum Teams weisen bereits im Rahmen des Planning Meetings die Aufgaben spezifischen Entwickler:innen zu. Dies führt jedoch dazu, dass bereits zu Beginn ein vordefinierter Pfad für die entsprechende Person entsteht, was die Eigenverantwortlichkeit einschränkt. Scrumban verzichtet auf die frühe

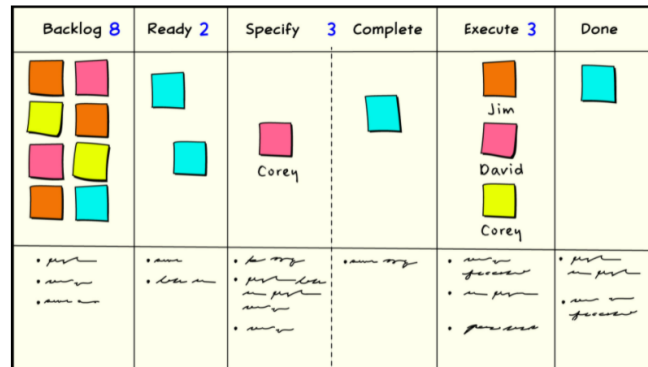


Abbildung 2.4: Beispiel eines Scrumban Boards [47]

Zuweisung und setzt wie auch Kanban auf einen Pull-Ansatz. Somit ist es in der Verantwortung der Entwickler:innen, nach Verfügbarkeit ein Item aus der Ready-Spalte zu übernehmen. Allgemein sollten keine Items direkt aus dem Backlog übernommen werden, sondern erst, wenn sie sich in der Ready-Spalte befinden. Um sicherzustellen, dass immer ausreichend Items in dieser Spalte sind, führt Scrumban einen Order-Punkt ein. Der Order-Punkt stellt eine Mindestanzahl an Tickets dar, die in der Ready-Spalte sein müssen. Wird dieser Punkt unterschritten, müssen die für die Planung verantwortlichen Personen dafür sorgen, dass neue Items geplant und in die Spalte gezogen werden. Somit erfolgt die Planung neuer User Stories immer erst nach Bedarf [26]. Allgemein gilt es bei Scrumban jedoch zu beachten, dass die Visualisierungspraktiken auf dem Board vor allem zur Arbeitsweise in der eigenen Organisation passen müssen und die Gestaltung somit variabel ist [33].

Die User Stories auf dem Scrumban Board werden wie bei Kanban durch Karten visualisiert. Die Art der Karte wird bei Scrumban farblich dargestellt. Das heißt für Bugs, Wartungsarbeiten oder auch neue Features werden verschiedene Farben zur leichteren Unterscheidung verwendet. Die Scrumban-Karten sollen ebenfalls das Start-, Fälligkeits- und Enddatum der Items darstellen. Das Enddatum stellt dabei das Datum dar, an dem das Item tatsächlich abgeschlossen wurde. Zusätzlich soll der geschätzte Zeitaufwand in Arbeitsstunden angegeben werden. Jedes Ticket sollte ebenfalls über eine Definition of Done verfügen, über welche festgestellt werden kann, ob die Karte tatsächlich abgeschlossen wurde. Außerdem sollte jede Spalte auf dem Scrumban Board festgelegte Bedingungen haben, die erfüllt werden müssen, damit die Karten in die nächste Spalte verschoben werden dürfen. Damit kann die Arbeitsqualität über den gesamten Entwicklungsprozess aufrecht erhalten werden. Zusätzlich zu den Bedingungen dürfen auch die WIP-Limits der nächsten Spalte noch nicht ausgelastet sein. Dies stellt sicher, dass nicht zu viele unfertige Items gleichzeitig auf dem Board vorhanden sind. Zusätzlich zu den Limits pro Spalte führt Scrumban auch WIP-Limits pro Person ein. Diese Limits, auch Multitasking Limits genannt, stellen sicher, dass die Entwickler:innen nicht an zu vielen Items gleichzeitig arbeiten, sondern ihr Fokus auf eine kleine beschränkte Anzahl an Items konzentrieren kann [33].

Scrumban selbst gibt keine spezifischen Rollen für die Teams vor. Somit können die in Scrum definierten Positionen weitestgehend übernommen werden. Auch die Funktion

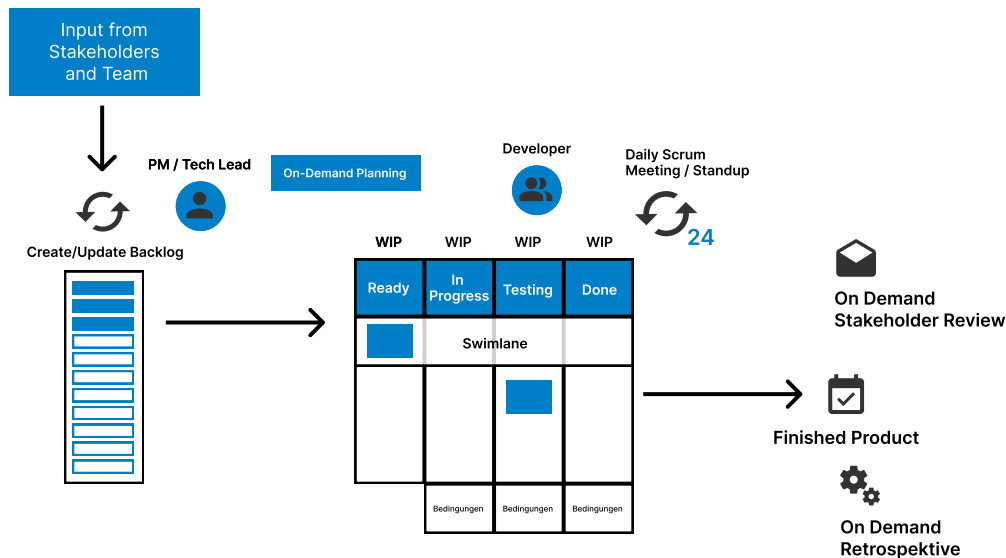


Abbildung 2.5: Darstellung eines Scrumban Prozesses

der Scrum Master:innen kann erhalten bleiben. In Scrumban sind sie für die Überwachung der Visualisierung und des kontinuierlichen Arbeitsflusses zuständig. Zusätzlich fällt die Überwachung von Maßnahmen, wie Anpassungen bei den WIP-Limits, und die Notwendigkeit von Events in ihren Aufgabenbereich. Im Scrumban Prozess, siehe Abbildung 2.5, können die Events des Daily Standups, des Sprint Reviews und der Sprint Retrospektive aus Scrum wiederverwendet werden. Da Scrumban nicht auf Sprints basiert, fokussieren das Review und die Retrospektive auf den Zeitraum zwischen den letzten beiden Events. Die Dauer zwischen den Events kann dabei von der jeweiligen Organisation selbst gewählt werden [33].

2.5 Probleme von etablierten agilen Prozessmodellen in Bezug auf Cyber Security

Um Herausforderungen von agilen Prozessmodellen im Zusammenhang mit Cyber Security ausfindig zu machen, müssen zuerst die Begriffe von Cyber Security und sicherer Softwareentwicklung definiert werden. Craigen, Diakun-Thibault und Purse definieren den Begriff Cyber Security als die Sammlung und Organisation von Strukturen, Ressourcen und Prozessen, die dazu beitragen, den Cyberspace, eine nicht real existierende Welt, vor unrechtmäßiger Beanspruchung von Eigentum zu schützen [11]. Die Beanspruchung bezieht sich dabei auf den Zugriff, die Entfernung, Verwaltung, Entfremdung und Extraktion von digitalem oder analogem Eigentum. Cyber Security stellt ein interdisziplinäres Feld dar, welches nicht auf nur einen Bereich oder ein System eingegrenzt werden kann. Rosenthal unterscheidet in seiner Definition von Cyber Security zusätzlich noch zwischen fünf Arten von Cyber Security [65]:

- Schutz kritischer Infrastruktur: Bezieht sich auf jene, die die Grundversorgung der Bevölkerung gewährleisten, wie zum Beispiel Kraftwerke für Strom und Wasser.
- Application Security: Fokussiert auf den Schutz von Anwendungen auf unterschiedlichste Weise.
- Netzwerk Sicherheit: Der Begriff beschreibt die Sicherung und Überwachung von internen Netzwerken einer Organisation.
- Cloud Sicherheit: Hierbei geht es um den Einsatz von Softwarelösungen, die die Daten und Ressourcen von Cloud Projekten schützen und überwachen.
- Internet of Things Security: Diese fokussiert auf den Schutz von physischen Geräten, die Zugang zum Internet aufweisen. Darunter fallen Geräte wie Router, Drucker oder auch Überwachungskameras.

Die konkreten Möglichkeiten der Beanspruchung von fremden Eigentum im Cyberspace sind dabei sehr vielfältig und von der entsprechenden Art des Cyber Security Einsatzgebiets abhängig. Das Open Web Application Security Project (OWASP) gibt jedes Jahr eine Liste der zehn relevantesten Sicherheitsrisiken (OWASP Top 10) heraus. Im Bereich der Application Security stellten dabei im Jahr 2021 fehlerhafte Access Controls², kryptografische Fehler³ und Injections⁴ die häufigsten Risiken dar [64].

Die Eingliederung von Sicherheitsmaßnahmen in agilen Prozessen stellt viele Organisationen vor Probleme. Eines der Probleme von agilen Prozessmodellen in Bezug auf die Integration von Cyber Security ist, dass die Anforderungen eines damit durchgeführten Softwareprojekts im Vorfeld nicht einschätzbar sind, was die Planbarkeit von Sicherheitsmaßnahmen erschwert [35]. Dies ist der Fall, da sich die Anforderungen im Laufe der Iterationen des Projekts wiederholt verändern können. Zusätzlich stehen bestimmte Security Praktiken wie ausführliche Security Reviews, zusätzliche Dokumentation von sicherheitsrelevanten Stellen im System oder auch detailliertes Security Testing im Widerspruch zu den Werten der Agilität, da diese die Flexibilität und Anpassungsfähigkeit der agilen Prozessmodelle einschränken würden [34]. Laut einer qualitativen Studie, die 16 amerikanische Organisationen aus unterschiedlichen Bereichen der Softwareentwicklung befragte, tendieren Organisationen bei agilen Prozessmodellen dazu, nicht-funktionale Anforderungen, wie Sicherheitsaspekte, zu ignorieren [32]. Um hohe Flexibilität, Effizienz und Schnelligkeit zu erreichen, verzichten Organisationen auf Prozesse, wie die Dokumentation und Ausführung von Security Maßnahmen, die nicht direkt dazu beitragen, das Produkt weiterzuentwickeln. Aufgrund der kurzen Entwicklungszyklen, zum Beispiel durch Sprints, bleibt nicht genug Zeit, um alle Schritte einer ausführlichen Security Planung, wie in traditionellen Prozessmodellen, durchzuführen. Dies führt jedoch dazu, dass das Risiko eines Securityvorfalls steigt [53]. Traditionellerweise werden Security Maßnahmen dadurch sequentiell geplant, was sich nur schwer mit dem iterativen Ansatz agiler Prozesse vereinbaren lässt [34].

²Access Control stellt sicher, dass User:innen nicht auf Teile eines Systems zugreifen können, welche außerhalb ihrer Berechtigungen liegen [64].

³Kryptografische Fehler stellen Fehler bei der Verschlüsselung von Daten während dem Austausch sowie der Speicherung der Daten dar [64].

⁴Injections sind Versuche, bei denen schadhafter Code in eine Anwendung eingeschleust wird [64].

Trotz der Herausforderungen und Probleme, die im Zusammenspiel von agilen Prozessen mit Cyber Security auftreten können, schließen sich die beiden Begriffe nicht gegenseitig aus. Aufgrund der Tatsache, dass die Berücksichtigung von Sicherheitsaspekten einen essentiellen Bestandteil von Softwarelösungen aus der Sicht von Kund:innen und der Organisation einnimmt, ist es von großer Relevanz, Security Praktiken in agile Prozessmodelle zu integrieren. Die Herausforderung dabei besteht darin, die Schritte aus der traditionellen Security Planung im Wasserfallmodell auf die Anforderungen von agilen Modellen anzupassen, ohne dass die Charakteristiken der Agilität darunter leiden [53]. Das folgende Kapitel 3 beschäftigt sich deshalb mit bestehenden Ansätzen der Integration von Security in agilen Prozessen.

Kapitel 3

Grundlagen bestehender Modelle zur Integration von Security in den Softwareentwicklungsprozess

Dieses Kapitel stellt bestehende Modelle zur Integration von Cyber Security in den Softwareentwicklungsprozess vor. Zu Beginn werden die Begriffe des Secure Software Development Lifecycles (S-SDL) und des Capability Maturity Modells (CMM) erläutert. Anschließend werden bestehende Modelle, die agilen sowie traditionellen Prozessen zugeordnet werden können, vorgestellt, analysiert und Gemeinsamkeiten hervorgehoben.

Der Begriff der sicheren Softwareentwicklung wird von McGraw durch die Ausführung von einer Sammlung an sicherheitsrelevanten Aktivitäten in Zusammenhang mit Softwareentwicklungsprozessen beschrieben [27]. Laut Microsoft setzt sich sichere Softwareentwicklung dabei aus drei Elementen zusammen, welche auf unterschiedliche Bereiche fokussieren: Security durch Best Practices, Prozessverbesserungen und Metriken, um die Auswirkungen der ersten beiden Elemente zu messen [28]. Oftmals wird die Durchführung dieser Aktivitäten im Rahmen eines Secure Software Development Lifecycles mittels eines Wasserfall-ähnlichen Prozessmodells durchgeführt. Wasserfallmodelle eignen sich für die Einführung von Security Maßnahmen gut, da hierbei bereits im Vorfeld die gesamte Planung hinsichtlich aller Anforderungen des Projekts durchgeführt wird. Dadurch können in allen folgenden Phasen die notwendigen Security Maßnahmen getroffen werden, da das Ziel des Projektes somit bereits vollständig definiert ist [35]. Jedoch existieren auch Ansätze zur Integration von Sicherheitsmaßnahmen in agilen Prozessen, siehe Abschnitt 3.2.

Ein Software Development Lifecycle (SDL) ist ein Framework, das beschreibt, wie Organisationen Softwareanwendungen von Beginn bis zum Ende entwickeln und fertigstellen. Es existiert eine Vielzahl an unterschiedlichen SDL-Varianten, jedoch bestehen die meisten aus den gleichen fünf Elementen beziehungsweise Phasen: Identifikation von Anforderungen, Softwarearchitektur und -design, Umsetzung, Testen und Veröffentlichung inklusive Wartung der Anwendung. Um sichere und zuverlässige Anwendungen zu erstellen, benötigt es die Integration von Security in allen fünf Phasen des Prozesses. Der daraus entstehende Lebenszyklus stellt nun ein Modell eines S-SDL dar [29]. Khan

und Zulkernine beschreiben den S-SDL als einen Prozess, der Security Methoden während des Lebenszyklus der Softwareentwicklung berücksichtigt. Security Methoden können dabei Anforderungserhebung mit Fokus auf Security oder auch Security Assurance Modelle ¹ enthalten [25]. Die Einführung eines Secure Software Development Prozesses stellt einen wichtigen Schritt für die Gewährleistung von sicheren Anwendungen dar. Oftmals wird davon ausgegangen, dass es ausreichend ist, im Rahmen der Testphase eines Prozesses auf Security Probleme zu überprüfen. Dieser Ansatz führt jedoch dazu, dass Fehler erst relativ spät gefunden werden können, da zu diesem Zeitpunkt der Entwicklungsprozess bereits stark fortgeschritten ist [8]. Je später Fehler entdeckt werden, desto höher fallen die Kosten für die Behebung des Fehlers aus. Ein Security Fehler der erst in der Testphase identifiziert wird, verursacht somit viel höhere Kosten als wenn der gleiche Fehler bereits während der Erhebung der Anforderungen erkannt wird [27].

Neben unterschiedlichen SDL-Modellen werden noch Capability Maturity Modelle (CMM) verwendet. Diese geben im Vergleich zu einem SDL keinen vollständig definierten Prozessablauf vor, sondern stellen ein Referenzmodell dar, welches weiterführende Praktiken enthält, um den Softwareentwicklungsprozess in einem bestimmten Teilbereich der Softwareentwicklung zu optimieren. Somit gibt ein CMM keinen Prozess sondern die Charakteristiken eines Prozesses vor. Der Zweck eines CMM ist, dass Organisationen ihre bestehenden Praktiken mit dem Referenzmodell vergleichen und darauf aufbauend ihre eigenen Prozesse optimieren können [13].

3.1 Methodische Vorgehensweise bei der Literaturanalyse

Für diese Arbeit wird eine integrative Literaturanalyse nach Cooper [10] eingesetzt. Diese stellt eine Form der Forschung dar, die repräsentative Literatur zu einem Thema in einer integrierten Sichtung, Kritik und Synthese zusammenfasst, wodurch neue Rahmenbedingungen und Perspektiven auf das Thema entstehen können. Hierfür werden sowohl kürzlich veröffentlichte Quellen, die den aktuellen Stand der Forschung repräsentieren, sowie auch ältere Publikationen berücksichtigt [44]. Der Prozess der integrativen Literaturanalyse besteht aus fünf aufbauenden Phasen. Im ersten Schritt werden die Problemstellungen der Arbeit, die Forschungsfragen, definiert. In der zweiten Phase wird Literatur gesammelt und der Fokus auf einen bestimmten Aspekt gelegt. Der dritte Schritt beschäftigt sich mit der Evaluation der gesammelten Quellen. Hierbei werden Faktoren, wie die Relevanz, der inhaltliche Zusammenhang zu den anderen gefundenen Quellen sowie die Zuverlässigkeit der Quellen berücksichtigt. Die vierte Phase stellt eine Analyse und Interpretation dar. Im Rahmen des letzten Schrittes werden die gefundenen Ergebnisse vorgestellt [10].

Die für diese Arbeit ausgewählte Literatur wurde anhand unterschiedlicher Kriterien evaluiert und ausgewählt. Um die Qualität sicherzustellen, werden vor allem wissenschaftliche Publikationen herangezogen. Primär jene Quellen, die Prozesse zur Qualitätssicherung, wie zum Beispiel ein Peer Review, durchlaufen haben. Zusätzlich wird darauf geachtet, dass die methodische Vorgehensweisen der Quellen nachvollziehbar und transparent festgehalten wurden, sowie dass die Publikationen häufig in ande-

¹Software Assurance Modelle stellen sicher, dass die ausgewählten Sicherheitsmechanismen von Systemen für eine Anwendung effektiv funktionieren [14]. Ein Beispiel für ein Software Assurance Modell ist OWASP SAMM [30].

ren wissenschaftlichen Werken referenziert werden, um deren Reichweite und Stellung in der Wissenschaft zu beurteilen. Als weiteres Merkmal wurde überprüft, ob die Herausgeber:innen und Autor:innen aus dem wissenschaftlichen Umfeld stammen. Um die Qualität von Online-Werken zu beurteilen, wurden die Herausgeber:innen der Website überprüft, wobei Plattformen von Organisationen und Institutionen bevorzugt wurden, die an den veröffentlichten Inhalten wenig kommerzielles Interesse hegen. Da es im für diese Arbeit relevanten Teilbereich der Cyber Security aufgrund des kontinuierlichen technischen Fortschritts zu häufigen Änderungen kommt, wird hierbei auch auf die Aktualität der ausgewählten Publikationen geachtet. Jedoch wird die Verwendung älterer Quellen nicht kategorisch ausgeschlossen, da manche Praktiken immer noch eine Relevanz für die Verbesserung der Cyber Security aufweisen.

Die Literaturrecherche wurde mittels einer Vielzahl an unterschiedlichen Plattformen durchgeführt. Primär wurden Literaturdatenbanken verwendet, die vom Campus Hagenberg der Fachhochschule Oberösterreich zur Verfügung gestellt werden, vorrangig die Datenbanken SpringerLink, IEE Xplore und ScienceDirect. Zusätzlich werden wissenschaftliche Publikationen über Google Scholar und Researchgate gesucht. Dafür werden in den Suchfunktionen der Plattformen Schlagwörter wie „Scrumban“, „agile Prozessmodelle“, „Software Development Lifecycle“, „agile Software Development Lifecycle“, „agile Secure Software Development Lifecycle“ oder „Cyber Security Best Practices“ in unterschiedlichen Kombinationen eingegeben. Alle verwendeten Schlagwörter wurden in deutscher und englischer Sprache gesucht, um die Anzahl an relevanten Ergebnissen zu erhöhen. Diese Schlagwörter werden ebenso in der Google Suche eingegeben um entsprechende Online-Quellen zu finden. Im ersten Schritt wird anhand des Abstracts der gefundenen Quellen festgestellt, ob die Publikation zur Beantwortung der Forschungsfragen beiträgt. Danach werden die Publikationen gelesen, die Quintessenz festgehalten und anhand der davor beschriebenen Kriterien auf ihre Qualität evaluiert und ausgewählt. Anschließend wird das Schneeballprinzip angewendet, um über die verwendeten Zitate weitere Quellen zu finden. Durch diese Vorgehensweise wurde eine Sammlung an qualitativen und relevanten Quellen aufgebaut und anschließend systematisch zusammengefasst, interpretiert und um neue Aspekte erweitert.

3.2 Bestehende Secure Software Development Modelle

Dieser Abschnitt analysiert bestehende CMM- und SDL-Modelle zur Berücksichtigung von Cyber Security Maßnahmen während dem Softwareentwicklungsprozess. Für die Gegenüberstellung wurden nur Modelle berücksichtigt, die von Organisationen oder im Rahmen von wissenschaftlichen Arbeiten publiziert wurden. Hierbei wird auf den Secure Development Lifecycle von Cisco [31], den Security Development Lifecycle von Microsoft [28], das Touchpoints-Modell von McGraw [27], die Secure Software Development Practices von SAFECODE [36], das Secure Software Development Framework des National Institute of Standards and Technology (NIST) [16], das Software Assurance Maturity Model von OWASP [30], das Building Security In Maturity Model (BSIMM) [7], den Security Software Development Lifecycle (SecSDM) [19], den Security Development Lifecycle for Agile von Microsoft [28] und den New Secure Software Development Life Cycle (NSSDL) für agile Methoden [29] eingegangen.

Die Analyse der Modelle, siehe Tabelle 3.1, gliedert sich in folgende sechs Kategorien:

- Agil: Diese Kategorie besagt, ob das analysierte Security Modell den traditionellen oder den agilen Prozessmodellen zuordenbar ist. Ein Modell gilt als agil, wenn es auf den Werten des agilen Manifests basiert oder auf etablierten agilen Prozessmodellen, wie beispielsweise Scrum, aufbaut.
- Prozess: Hierbei werden die empfohlenen Prozessschritte der Modelle beschrieben.
- Methoden: In dieser Kategorie werden die entsprechenden Security Methoden und Security Best Practices der Modelle angeführt.
- Events: Die Events-Kategorie gibt an, ob das jeweilige Modell spezifische Security-Events oder Security-Meetings für den Softwareentwicklungsprozess vorschlägt.
- Rollen: Diese Kategorie gibt an, ob die Modelle spezielle Security Rollen und Positionen für die Durchführung des Prozesses vorgeben.
- Praxis: Diese Kategorie beschreibt, ob das entsprechende Modell in der Praxis von Unternehmen tatsächlich eingesetzt wird. Ein Modell gilt als in der Praxis eingesetzt, sofern es von einer Organisation für den internen Einsatz publiziert wurde. Zusätzlich wird im Zuge einer Online Recherche überprüft, ob Berichte über den Einsatz des entsprechenden Modells vorliegen.

Zur Beantwortung der Kategorien wird neben der textuellen Auflistung der entsprechenden Praktiken auch eine dreistufige Skala bestehend aus Häkchen (✓), Welle (∼) und Kreuz (×) verwendet. Das Häkchen drückt dabei aus, dass das entsprechende Modell die jeweilige Kategorie erfüllt. Das Kreuz besagt, dass die jeweilige Kategorie nicht erfüllt wurde oder dass keine Praktiken aus dieser Kategorie im Modell angeführt werden. Die Welle signalisiert, dass es keine Belege für die jeweilige Kategorie in dem entsprechenden Modell gibt.

3.3 Erkenntnisse aus bestehenden Modellen

Auf Basis der Ergebnisse der durchgeführten Analyse, siehe Tabelle 3.1, lässt sich erkennen, dass die vorgestellten Modelle, die auf traditionellen Prozessmodellen beruhen, allesamt aus einem ähnlichen Ablauf bestehen. Dabei werden zu Beginn Security Aspekte in der Planung sowie der Erhebung der Anforderungen berücksichtigt. Anschließend werden Secure Design Praktiken vorgeschlagen. Auf das Secure Design folgen die Phasen der Implementierung sowie des Testings. Abschließend wird in einer Phase darauf eingegangen, wie mit gefundenen Fehlern umgegangen wird. Bei den agilen Modellen werden ausgewählte Praktiken in den Entwicklungsprozess inkludiert. Aufgrund der Tatsache, dass bei den agilen Prozessen iterativ alle Phasen kontinuierlich durchlaufen werden, gibt es keine strikte Abgrenzung zwischen den einzelnen Phasen. Die beiden vorgestellten agilen Modelle, NSSL [29] und Microsoft SSDL for Agile [28], unterscheiden sich jedoch bei der Gliederung des Prozesses. Microsoft unterscheidet zwischen Security Praktiken die einmal, immer oder manchmal ausgeführt werden müssen. Somit kann eine höhere Anzahl an unterschiedlichen Praktiken innerhalb der kurzen Releasezyklen durchgeführt werden. Der NSSL nimmt dagegen diese Unterscheidung nicht vor und orientiert sich an der Vorgehensweise des Kanbanprozesses. Somit werden alle Praktiken ständig in jeder Iteration durchgeführt. Die beiden agilen Ansätze setzen auch auf

3. Grundlagen bestehender Modelle zur Integration von Security in den Softwareentwicklungsprozess

18

Tabelle 3.1: Analyse bestehender Security Modelle

Security Modell	Agil	Prozess	Methoden	Events	Rollen	Praxis
Cisec Secure Development Lifecycle	X	1. Plan	Threat Modeling Privacy Assessment	X	X	✓
		2. Develop	Secure Code Repositories Static Code Analysis Security Training			
		3. Validate	Vulnerability Testing SCA Testing			
		4. Launch	X			
		5. Operate	Penetration Testing			
		6. Monitor	Logging			
		0. Training	Security Training			
Microsoft SSDL	X	1. Requirements	Security Requirements Security Bug Bar	Security Push Final Security Review	Security Advisor	✓
		2. Design	Secure Design Requirements Threat Modeling			
		3. Implementation	Implementation Best Practices Static Code Analysis			
		4. Verification	Dynamic Code Analyse Fuzz Testing			
		5. Release	Security Review			
		6. Response	Incident Response Plan			
			Abuse Cases Security Requirements Risk Analysis			
Touchpoints von McGraw	X	1. Requirements and Use Cases	Risk Analysis	X	X	~
		2. Architecture and Design	Risk Analysis			
		3. Test Plans	Security Testing			
		4. Code	Code Review (Tools)			
		5. Tests and Test Results	Risk Analysis Penetration Testing			
		6. Feedback from the Field	Penetration Testing Security Operations			
			Secure Design Principles Threat Modeling Access Management			
SAFECode Secure Software Development Practices	X	1. Design	Secure Coding Standards	X	X	~
		2. Secure Coding	(Automated) Security Testing Software Composition Analysis			
		3. Testing and Validation	Define Severity Risk Acceptance Criteria			
		4. Manage Security Findings				
NIST Secure Software Development Framework	X	1. Prepare the Organisation	Security Requirements Security Rollen und Training Security Practices Security Tools Security Akzeptanzkriterien Secure Development Environment	Security Review Security Retrospective	allgemeine Cyber Security Rollen	~
		2. Protect Software	Access Protection Archive Software Releases			
		3. Produce Well-Secured Software	Security Requirements Threat Modeling Reuse Secure Software Security Testing Secure Coding Practices			
		4. Respond to Vulnerabilities	Monitor Vulnerability Databases Incident Response Plan Vulnerabilities Prioritization and Analysis			
			Security Metrics and Strategy Security Policy Security Training			
		2. Design	Threat Modeling Security Requirements Secure Design / Architecture			
		3. Implementation	SAST, DAST, SCA Testing Secure Deployment Track Security Defects			
OWASP SAMM	X	4. Verification	Security Testing (Fuzzing, Unit Tests, Penetration Testing)	X	Security Champion	✓
		5. Operations	Incident Response Plan Environment / Operational Management			
		1. Governance	Security Strategy / Metrics Security Requirements Security Compliance and Policy Security Training			
		2. Intelligence	Investigate Possible Attacks Abuse Cases Secure Design Security Standards Secure Coding Standards			
		3. SSDL Touchpoints	Automated Tools Security Testing			
BSIMM	X	4. Deployment	Penetration Testing Application Monitoring Incident Response Plan Bug Management	Security Review	Security Executives (e.g. CISO) Software Security Group Satellite	✓
			Threat/Risk Identification Risk Prioritisation			
		2. Analysis Phase	Risk Analysis			
		3. Design Phase	Auswahl von Security Maßnahmen			
SecSDM	X	4. Implementation	Security Testing Software Security Tools	X	X	X
		5. Maintenance Phase	Dokumentation			
		One-Time Requirements	Basis Threat Model			
		Every Sprint Requirement	Code Analyse Tools Threat Modelling (neue Features) Security Training Software Composition Analysis			
		Bucket Requirements - Verification Tasks - Design Review - Planning	Fuzz Testing Statistische Code Analyse Software Composition Analyse Security Bugs Klassifizierung Incident Response Plan erstellen			
New Secure Software Development Life Cycle	✓	Kanban Prozess	Security Backlog Pair Programming Security Training Risk Analysis / Threat Modeling Security Testing Secure Implementation Practices	Berücksichtigung von Security Faktoren in allen agilen Meetings	Security Master Security Gurus	~

unterschiedliche agile Prozessmodelle. Der Prozess von Microsoft stellt eine Erweiterung des Scrum Prozesses dar, wohingegen der NSSDL auf einem Kanbanprozess basiert.

Allgemein setzen alle vorgestellten Modelle auf das Prinzip des Pushing Left, auch Shifting Left genannt. Das Prinzip besagt, dass Security Maßnahmen so früh wie möglich in den Entwicklungsprozess inkludiert werden sollten. Dies ermöglicht es, Security-Probleme früher im Prozess zu entdecken und somit Kosten zu verhindern [24].

Auch bei den eingesetzten Methoden lassen sich ähnliche Ansätze zwischen den unterschiedlichen Modellen beobachten. Sieben von insgesamt zehn vorgestellten Modellen inkludieren das Training der Entwickler:innen als wichtigen Schritt in den Prozess. Im Bereich der Erhebung von Security Anforderungen und der Analyse von möglichen Risiken wird das Konzept des Threat Modeling, siehe Abschnitt 4.2.2, in neun Modellen vorgeschlagen. Zusätzlich sollten Security Requirements (n=6), die Klassifizierung von Security-Problemen (n=3), Abuse Cases (n=2) und Secure Design Prinzipien (n=4) berücksichtigt werden. Im Zuge des Entwicklungs- und Testing-Prozesses empfehlen alle zehn Modelle, dass Security Testing Methoden eingesetzt werden. Dabei werden Software Composition Analysis, Static Code Analysis, Dynamic Code Analysis, zum Beispiel Fuzzing, Unit Testing und Penetration Testing vorgeschlagen. Zusätzlich sollten auch Secure Coding Praktiken (n=4) während der Entwicklung berücksichtigt werden. Für die Zeit nach dem Deployment von Software schlagen fünf Modelle die Erstellung eines Incident Response Plans² vor.

Kapitel 4 stellt ausgewählte Praktiken näher vor, die auf Basis der oben beschriebenen Analyse identifiziert wurden. Bei der Auswahl wurde darauf geachtet, dass die Praktiken mit der Vorgehensweise des Scrumban Prozesses sowie der agilen Softwareentwicklung kompatibel sind. Zusätzlich werden nur jene Praktiken vorgestellt, die im Zuge der Analyse in mehreren bestehenden Modellen gefunden wurden. Auch der mögliche Einsatz der Praktiken innerhalb von Scrumban wird dargelegt.

²Ein Incident Response Plan stellt die Dokumentation von einer Sammlung an Instruktionen dar, die angewandt werden, falls die Organisation Opfer einer schädlichen Cyberattacke wird [43].

Kapitel 4

Integration von Security Maßnahmen in den Scrumban Prozess

Nach den Definitionen und Grundlagen über agile Prozessmodelle und Cyber Security in Kapitel 2 sowie der Analyse von bestehenden Secure Software Development Modellen in Kapitel 3, widmet sich dieses Kapitel der Beantwortung der in Kapitel 1 vorgestellten Forschungsfragen. Im Folgenden werden die auf Basis der durchgeführten Literaturrecherche identifizierten Best Practices angeführt und deren Einsatz in einem agilen Scrumban Prozess aufgezeigt. Außerdem werden Tools vorgestellt, die ebenfalls unterstützend dazu beitragen, den Scrumban Prozess sicherer zu gestalten.

4.1 Der menschliche Faktor

Einer der wichtigsten Faktoren, um Cyber Security Maßnahmen in den agilen Softwareentwicklungsprozess zu inkludieren, sind die Teilnehmer:innen des Prozesses selbst. Damit eine Organisation erfolgreich einen agilen S-SDL einführen und umsetzen kann, muss die gesamte Organisation auf die Wichtigkeit von Cyber Security aufmerksam gemacht werden. Wenn Individuen nicht verstehen, weshalb sie bestimmte Maßnahmen umsetzen sollten und welche Auswirkungen die Nichtdurchführung dieser haben könnten, sinkt die Wahrscheinlichkeit, dass die Personen die Maßnahmen unterstützen und somit auch effektiv durchführen [36].

4.1.1 Security Training

Um die Security Kompetenz innerhalb der Organisation zu erhöhen, empfiehlt es sich Security Trainings in der gesamten Organisation einzuführen, da viele Entwickler:innen nur über ein geringes Wissen an Security Praktiken verfügen. Hierfür sollten sich jedoch nicht nur die Mitglieder:innen des Scrumban Teams dem Training unterziehen, sondern auch alle anderen Personen innerhalb der Organisation, damit auch den restlichen Mitarbeiter:innen die Bedeutung von Sicherheitsaspekten bewusst wird. Für Entwickler:innen und Tester:innen ist ein vertiefendes Training sinnvoll [31]. Die Aufbereitung der Cyber Security Trainingsinhalte kann dabei auf unterschiedliche Arten erfolgen. Hierbei bieten sich Formate, wie Trainings mit physischen Vortragenden, die entweder

aus der eigenen Organisation oder von externen Firmen stammen, oder auch computerunterstützte E-Learning Programme, wie zum Beispiel Lernvideos oder Web Based Trainings (WBT), an [30].

Um sicherzustellen, dass alle Mitarbeiter:innen die Grundlagen von Cyber Security verstehen, sollten bereits im Onboarding Prozess für neue Angestellte verpflichtende Cyber Security Module durchgemacht werden. Allgemeine Themen, die sich hierfür anbieten, sind beispielsweise Passwortsicherheit oder Cyber Security im regulären Arbeitsalltag. Das Ziel ist dabei das Erlernen der Sicherheitskultur des Unternehmen. Der Onboarding Prozess stellt jedoch nur den ersten Schritt des Security Trainings dar und ersetzt ein detailliertes Security Training nicht. Das Cyber Security Trainingsprogramm sollte dabei aus einem verpflichtenden Grundkurs bestehen, zum Beispiel „Einführung in Software Security“, welcher von allen technischen Positionen absolviert werden muss. Der Inhalt des Grundkurs ist hierbei noch nicht auf spezielle Anforderungen zugeschnitten. Dies ist relevant, da das Erlernen von vertiefenden Cyber Security Maßnahmen direkt zu Beginn demotivierend auf die Lernenden wirkt. Deshalb sollten anschließend weiterführende Kurse oder Module zur Verfügung gestellt werden, die auf die bestimmten Rollen, eingesetzten Technologien und verwendeten Programmiersprachen der Mitarbeiter:innen zugeschnitten werden [7]. Somit könnte es beispielsweise ein Programm für Frontend Entwickler:innen, für Backend Entwickler:innen, für Tester:innen aber auch ein Modul für Programmiersprachen wie Ruby, Kotlin, JavaScript oder auch PHP geben. Das Ziel der Security Trainingsprogramme ist, den Wissensstand der in einem agilen S-SDL Modell teilnehmenden Personen bestmöglich zu erweitern [31].

Ein weiterer wichtiger Aspekt des Security Trainings ist die Regelmäßigkeit der Durchführung. Durch die Einführung von online verfügbaren Ressourcen mittels WBTs oder Videokursen können Personen zeit- und ortsunabhängig auf die Trainingsressourcen zugreifen. Neben den ständig abrufbaren Ressourcen ist es jedoch notwendig einen verpflichtenden jährlichen Cyber Security Auffrischkurs abzuhalten. Die Auffrischung stellt sicher, dass das bereits erlernte Wissen der Mitarbeiter:innen immer noch aktuell ist. Der Kurs kann dabei in einem die ganze Organisation betreffenden Security Tag durchgeführt werden [7]. Essentiell ist, dass die verwendeten Ressourcen laufend auf ihre Aktualität überprüft und gegebenenfalls adaptiert oder erweitert werden [16]. Um die durch das Security Training entstandenen Anforderungen und Aufgaben zu erfüllen, sollte das Scrumban Team zusätzlich durch neue Positionen und Rollen ergänzt werden, siehe Abschnitt 4.1.2.

4.1.2 Security Rollen

Zusätzlich zu dem Security Training sollten auch neue Rollen das Scrumban Team ergänzen, da die Einführung von Security Maßnahmen sowie auch die Gestaltung des Cyber Security Trainings Expert:innenwissen benötigt, welches potentiell in dieser Form im bestehenden Team nicht gegeben ist [36]. Es empfiehlt sich, die Rolle des oder der Security Master:in in den Prozess einzuführen. Diese Person verfügt über detailliertes Expert:innenwissen über Cyber Security und ist dafür verantwortlich, alle sicherheitsrelevanten Entscheidungen, die während der Abhaltung des Softwareentwicklungsprozesses nötig sind, zu treffen. Zusätzlich dient diese Person als Kontakt für alle Teammitglieder:innen, um Fragen über sicherheitsrelevante Aspekte zu klären. Um detailliertes

Security Wissen in spezifischen Domänen in der Organisation zu sammeln, empfiehlt sich die Einführung einer interdisziplinären Security Gruppe [29]. Die Größe und Zusammenstellung der Gruppe ist dabei von den Bedürfnissen der jeweiligen Organisation abhängig. Die Gruppe sollte jedoch zu einem Teil aus Personen bestehen, die selbst in der Lage sind, Programmcode zu schreiben, um Security Code Reviews durchzuführen. Da auch gute Softwareentwickler:innen oftmals nicht auch noch gute Softwarearchitekt:innen sind, ist es ratsam, auch Softwarearchitekt:innen in die Security Gruppe aufzunehmen, um auch diesen Aspekt der Softwareentwicklung abzudecken [7]. Weitere Expertisenbereiche können je nach Bedarf in Netzwerksicherheit, Autorisierungen, Verschlüsselung und Development Operations (DevOps) zu der Gruppe hinzugefügt werden [29]. Die Hauptaufgabe dieser Gruppe besteht darin, anderen Entwickler:innen bei Security Problemen zu helfen und eine Kontaktmöglichkeit für spezifische Security Fragen zu bieten, die der oder die Security Master:in nicht vollständig beantworten kann [7].

4.2 Security durch den Backlog

Wie Kapitel 2 erläutert, basieren viele agile Prozessmodelle auf dem Prinzip eines Backlogs, der die Anforderungen eines Projektes enthält. Wie die Analyse bestehender Security Modelle in Kapitel 3 aufzeigt, wird in den meisten bestehenden Prozessmodellen auf das Pushing-Left-Prinzip gesetzt. Somit sollten bereits vor Beginn des Entwicklungsprozesses Security Maßnahmen berücksichtigt werden. Bei agilen Prozessmodellen ist es nicht möglich zu Beginn alle möglichen Security Schwachstellen des zu entwickelnden Systems aufgrund sich ändernder Anforderungen zu identifizieren [38].

Security Aspekte müssen bereits im Product Backlog des Scrumban Prozesses beachtet werden. Hierbei ist es wichtig, dass die Sicherheitsmaßnahmen direkt im Backlog vorhanden sind und nicht in davon unabhängigen Anforderungslisten angeführt werden. Ansonsten besteht die Gefahr, dass die Items oder User Stories im Backlog rein anhand der funktionalen Anforderungen durch den oder die Product Owner:in gereiht werden und nicht-funktionale Anforderungen, wie die Cyber Security, nicht berücksichtigt werden [29]. Der Backlog besteht somit aus einer Sammlung von funktionalen User Stories, nicht-funktionalen User Stories sowie User Stories, für die beide Anforderungsarten relevant sind [38].

4.2.1 Security User Stories

In agilen Prozessmodellen, wie auch in Scrumban, werden Anforderungen in Form von User Stories oder Epics, weniger genau spezifizierte und somit umfangreichere Stories, definiert. User Stories werden dabei allesamt nach dem gleichen Muster formuliert: *Ich als Rolle XY möchte, dass Feature XY umgesetzt wird, da ich Grund XY habe*. Ein konkretes Beispiel für dieses Muster wäre: *Ich als User:in möchte, dass ich meine aktiven Abonnements einsehen kann, damit ich einen Überblick über meine Ausgaben bekomme*. In einem weiteren Schritt werden die User Stories je nach Größe in kleinere Aufgaben unterteilt, die sobald sie in die Ready-Spalte auf dem Scrumban Board verschoben werden, bereit zur Umsetzung durch die Entwickler:innen sind [38].

Das Konzept von Security User Stories trägt dazu bei, Security Anforderungen in Form von User Stories abzubilden und die sogenannte Security Debt eines Projektes zu

verringern. Für die Erstellung einer solchen Story müssen die Security Anforderungen für das zu entwickelnde Feature oder das neue Projekt berücksichtigt werden, siehe Abschnitt 4.2.2 [38]. Die Erstellung wird dabei im Rahmen des Scrumban Prozesses durch den oder die Product Owner:in vorgenommen. Da der oder die Product Owner:in potentiell nicht über das notwendige Wissen über Cyber Security für die ordnungsgemäße Erstellung verfügt, können die in Abschnitt 4.1.2 beschriebenen Rollen des oder der Security Master:in sowie Mitglieder:innen der Security Gruppe bei der Erstellung zu Hilfe gezogen werden.

Security Debt und Priorisierung

Die Security Debt beschreibt noch nicht abgeschlossene Backlog Items mit Relevanz für die Security. Die Ansammlung von mehreren nicht durchgeführten oder übersprungenen Security Items führt zu einer Security Schuld (engl. debt), was in weiterer Folge zu einer unsicheren Anwendung führt. Aus diesem Grund ist es wichtig, keine Security Debt entstehen zu lassen. Die beste Möglichkeit um dies zu verhindern, ist bei der Priorisierung der Items des Backlogs darauf zu achten, welche User Stories den größten Return on Investment (ROI) für das Projekt bieten. Dabei sollten funktionale und nicht-funktionale Aufgaben gleichermaßen beachtet werden. Auch wenn Aufgaben, wie die Weiterentwicklung oder der Aufbau eines Threat Models, siehe Abschnitt 4.2.2, für das Projekt keinen sofortigen ROI erzeugen, ist es trotzdem relevant, Aufgaben wie diese bei der Priorisierung zu berücksichtigen, da der ROI hierbei erst langfristig erkenntlich wird. Ebenso sollte bei der Priorisierung von Security Maßnahmen die Art der Anwendung, die verwendeten Programmiersprachen, das Deployment und andere Rahmenbedingungen beachtet werden. Je nach System haben bestimmte Security Aufgaben eine höhere Auswirkung auf die Reduzierung von Angriffsbereichen als andere. Dies bedeutet zwar nicht, dass die anderen Security Aufgaben unwichtig sind, jedoch sollte dies bei der Priorisierung beachtet werden [38]. Auch bei der Priorisierung kann der oder die Product Owner:in durch die eingeführten Security Rollen unterstützt werden.

Anwendungsbeispiel

Bei der Formulierung von Security User Stories wird auch das Quality Assurance (QA) Team einer Organisation berücksichtigt. Ein Beispiel für eine konkrete Security User Story wäre: *Als Entwickler:in möchte ich sicherstellen und als QA Teammitglied verifizieren, dass das System keine potentiell gefährlichen Funktionsaufrufe verwendet.* Diese User Story kann nun in weitere Backlog Items heruntergebrochen werden. Eines der Items wäre dabei der Einsatz von statischer Code Analyse, siehe Abschnitt 4.3.2, um bestimmte unsichere Funktionsaufrufe der jeweiligen Programmiersprache zu entdecken und anschließend mit einer sicheren Variante zu ersetzen. Darunter fallen beispielsweise die `eval()`-Funktion¹, die `exec()`-Funktion², oder auch die Funktionen `memcpy()` und

¹Die `eval()`-Funktion in PHP und JavaScript erlaubt, dass ein String als Code ausgeführt wird. Dies ist sehr gefährlich, da somit schadhafter Code, der eingeschleust wurde, auch ausgeführt werden kann [60].

²Über die `exec()`-Funktion in PHP können externe Programme ausgeführt werden [21].

`strcpy()`³ aus C und C++. Ein weiteres Item für diese User Story könnte die Durchführung von Fuzz Testing in regelmäßigen Zeitabständen sein [38]. Fuzz Testing ist eine Art des Black Box Testings, welche darauf abzielt, Fehler in der Implementierung durch das automatisierte Absenden von Daten auf Schnittstellen zu finden, ohne den Code selbst zu analysieren [63].

4.2.2 Security Anforderungen

Um die benötigten Security User Stories zu erstellen, müssen bei der Planung von neuen Funktionalitäten oder Projekten auch die Security Anforderungen erhoben und berücksichtigt werden, um diese als Story darstellen zu können. Dafür sollten Secure Design Prinzipien, Risk Identifikation und auch Threat Modeling berücksichtigt werden [36].

Secure Design Prinzipien

Für das Secure Design wurden von Saltzer und Schroeder acht Prinzipien festgelegt, die beachtet werden sollten, um sichere Systeme zu entwickeln: [40]

- Economy of mechanism: Das Design des Systems sollte so einfach und klein wie möglich gehalten werden.
- Fail-safe defaults: User:innen sollten von Beginn an keine Rechte für das System haben. Diese sollten ihnen erst bei Bedarf zugewiesen werden.
- Complete mediation: Alle Zugänge zu Objekten des Systems müssen explizit autorisiert werden.
- Least privilege: Kein Programm und keine User:innen sollten mehr Rechte haben als notwendig sind, um die geforderten Aufgaben zu erledigen. Somit hat niemand Rechte, die nicht für den jeweiligen Account nötig sind.
- Separation of Privilege: Ein Schutzmechanismus sollte eingeführt werden, der darauf basiert, dass zwei Schlüssel benötigt werden, um auf geschützte Informationen zuzugreifen. Wenn das System gesperrt ist, können die beiden Schlüssel physisch getrennt werden, wodurch ein Schlüssel nicht ausreichend ist, um auf das System oder die gesperrten Informationen zuzugreifen.
- Open Design: Es ist nicht notwendig, das Design eines Systems geheim zu halten, um es vor Angriffen zu schützen. Anstelle der Geheimhaltung sollte auf geschützte Schlüssel und Passwörter gesetzt werden.
- Least common mechanism: Mechanismen die unter mehreren User:innen geteilt werden und von denen alle Nutzer:innen abhängig sind, sollten vermieden werden. Geteilte Mechanismen stellen einen Informationspfad zwischen User:innen dar, welcher potentiell unabsichtliche Informationen über die beteiligten User:innen verraten könnte.

³Die `memcpy()`- und `strcpy()`-Funktionen schreiben in den Zwischenspeicher (Buffer) ohne vorher zu überprüfen, ob die Speichermenge innerhalb der zur Verfügung stehenden Grenzen liegt. Dies kann ausgenutzt werden, um zu große Daten in den Zwischenspeicher zu speichern. Dies führt in weiterer Folge zu einem sogenannten Buffer Overflow, wobei durch die zu große Speichermenge andere Daten im Speicher überschrieben werden [17].

- Psychological acceptability: Bei dem Design des System sollte auf eine leicht bedienbare Benutzer:innenoberfläche geachtet werden. Dies ist notwendig, damit die User:innen einfach und routinemäßig Schutzmechanismen in dem System durchführen können, ohne dass sie sich an eine spezielle neue Benutzer:innenoberfläche gewöhnen müssen.

Diese ursprünglichen acht Prinzipien wurden durch das Software Assurance Forum for Excellence in Code (SAFECode) noch um drei weitere Prinzipien für das Secure Design ergänzt: [36]

- Defense in depth: Das System sollte so designet werden, dass selbst wenn eine Schwachstelle (Vulnerability) gefunden oder ein Sicherheitsmechanismus überwunden wurde, das System dem Angriff immer noch standhalten kann. Für die Umsetzung bedeutet dies, dass mehrere Schichten an Security Maßnahmen benötigt werden, oder dass das System kontrolliert zusammenbricht sobald eine Security Maßnahme überwunden wurde, um zu verhindern, dass Angreifer:innen die Kontrolle über das gesamte System erlangen.
- Fail securely: Das System sollte sich selbst im Fehlerfall oder im Falle eines Systemabsturzes immer noch in einem sicheren Zustand befinden.
- Design for updating: Das System oder neue Features sollten so konzipiert werden, dass sie zu einem späteren Zeitpunkt jederzeit erneuert oder aktualisiert werden können. Dies ist notwendig, da sich auch die Security Anforderungen eines Systems jederzeit verändern können.

Die angeführten Prinzipien des Secure Designs vollständig im Zuge der Erstellung von Security User Stories zu erfüllen, stellt eine Herausforderung dar. Nichtsdestotrotz trägt die Berücksichtigung der Prinzipien bei der Planung von neuen Backlog Items dazu bei, sicherere Systeme zu erstellen. Die beschriebenen Prinzipien sind sehr allgemein gehalten und können somit auf unterschiedlichste Systeme angewandt werden. Die konkrete Umsetzung der Prinzipien ist für jedes System unterschiedlich und muss somit dementsprechend in dem jeweiligen Kontext angewandt werden [36].

Risiko Identifikation

Neben den Secure Design Prinzipien sollten bei der Planung von neuen Backlog Items auch potentielle Security Risiken der geplanten funktionalen Anforderungen identifiziert werden. Hierfür kann der Einsatz von Checklisten helfen um Risiken einfacher zu identifizieren [19]. Eine Möglichkeit dafür ist die Berücksichtigung der OWASP Top 10 Liste, die die zehn häufigsten Security Risiken enthält [64]. Es ist jedoch wichtig zu beachten, dass diese Liste nicht als vollständig und erschöpfend betrachtet wird, da weniger häufige Security-Probleme nicht berücksichtigt werden. Um auch weitere Risiken für die geplanten Features im Backlog zu identifizieren, empfiehlt sich auch hierbei Rücksprache mit den für die Security verantwortlichen Rollen innerhalb des Scrumban Teams zu halten. Die dabei getroffenen Entscheidungen sollten in schriftlicher Form innerhalb des zu erstellenden Backlog Items dokumentiert werden, um die Nachvollziehbarkeit zu ge-

währleisten. Um Beratungen mit den Security Rollen in Zukunft seltener zu benötigen, empfiehlt es sich, die getroffenen Entscheidungen zusätzlich in einer Wissensdatenbank zu dokumentieren [28]. Obwohl agile Prozessmodelle grundsätzlich eher versuchen unnötige Dokumentationen zu vermeiden, bietet die Entwicklung von Security Standards einen Mehrwert für die gesamte Organisation. Das Ziel der Dokumentation ist es, Standards für Probleme zu entwickeln, die bereits in der Vergangenheit aufgetreten sind. Die Standards erklären dabei akzeptierte Lösungswege für die Probleme. Beispielsweise könnte ein Standard beschreiben, wie Authentifizierung in einem System so sicher wie möglich umgesetzt werden kann. Hierbei können im Zuge des Standards bereits umgesetzte Backlog Items mit Referenz zu deren Umsetzung in der Dokumentation verlinkt werden, um praktische Beispiele zur besseren Orientierung zu liefern [7].

Threat Modeling

Ein weiterer wichtiger Aspekt um die Anforderungen für sichere Systeme zu erheben, ist die Erstellung eines Threat Models. SAFECode beschreibt die Durchführung eines Threat Models als eine der Security Aktivitäten mit dem höchsten ROI, um Designfehler zu erheben noch bevor die entsprechenden Features im Code umgesetzt werden [36].

Threat Modeling beschreibt den Prozess der Identifikation von potentiellen Bedrohungen und Risiken eines Systems und stellt sicher, dass Maßnahmen gegen die identifizierten Risiken vorhanden sind. In seiner einfachsten Form stellt Threat Modeling somit eine Art des Brainstormings dar, in dem alle möglichen Bedrohungen für eine Anwendung, ein System oder ein Produkt gesammelt werden. Man versetzt sich dabei gedanklich in die Rolle von potentiellen Angreifer:innen, um die Schwachstellen des Systems zu entdecken [24]. Dabei fokussiert man auf unterschiedliche Themenbereiche, wie Authentisierung, Verschlüsselung von Daten oder Validierungen, mit dem Ziel die Eintrittswahrscheinlichkeit der möglichen Schwachstellen so gering wie möglich zu halten. Man versucht somit bereits innerhalb einer frühen Designphase der Software Angriffspunkte wie Injection-Schwachstellen oder auch unverschlüsselte Daten zu verhindern. Auf Basis der entdeckten Security-Probleme lassen sich Anforderungen an das zu entwickelnde System ableiten [66].

Um Threat Modeling auch in einem agilen Scrumban Prozess effektiv einzusetzen, muss die Vorgehensweise bei der Erstellung angepasst werden. Da am Anfang noch nicht alle benötigten Anforderungen definiert werden können, müssen auch die laufend hinzugefügten Anforderungen mittels dem erstellten Threat Model abgebildet werden [66]. Im Falle der Erstellung eines neuen Produktes oder Systems sowie für den Fall, dass ein agiler Scrumban Prozess in ein vorher traditionell geführtes Projekt eingeführt wird, sollte ein Basis Threat Model erstellt werden. Bei der Erstellung des Basismodells sollte jedoch darauf geachtet werden, dass nur jene Teile und Features des Systems modelliert werden, die bereits schon existieren oder sich gerade in der Entwicklung befinden. Somit kann verhindert werden, dass die initiale Modellierung zu viel Zeit in Anspruch nimmt und der Planungsaufwand erheblich zunimmt. Die Erstellung des Basismodells für das System ist jedoch nicht ausreichend, um auch Risiken zu identifizieren, die durch die Einführung von neuen Features entstehen. Somit ist es notwendig, dass das Threat Model iterativ im Laufe des Prozesses angepasst wird. Jedes Mal bevor neue Features, die die aktuell bestehende Variante des Threat Models verändern würden, in die Ready-

Spalte auf dem Scrumban Board verschoben werden, muss das Modell angepasst werden. Aufgrund der Tatsache, dass die Erstellung eines Threat Models nicht einfach automatisierbar ist und somit bei jeder Adaption die Mitglieder:innen des Scrumban Teams benötigt werden, sollten im Zuge der Adaption nur jene neuen Features berücksichtigt werden, die in die Ready-Spalte verschoben werden sollten [28].

Die Durchführung des Threat Modelings kann auf mehrere Arten erfolgen. Basierend auf dem Threat Modeling Manifesto sollten dabei jedoch die folgenden Fragen behandelt werden:

- Woran wird gerade gearbeitet?
- Was könnte dabei nicht funktionieren?
- Was werden wir tun um dies zu verhindern?
- Haben wir die Arbeit gut genug durchgeführt [52]?

Um die erste Frage zu beantworten empfiehlt sich die Erstellung eines Modells, das die Architektur des Systems beschreibt. Für die zweite Fragestellung werden mögliche interne und externe Angriffsmöglichkeiten auf das erstellte Modell angewandt. Zur Klärung der dritten Frage werden auf Basis der Ergebnisse der vorherigen Fragestellungen Security Requirements erstellt, beispielsweise in Form von Security User Stories, die in den Product Backlog übernommen werden. Die vierte Frage kann als Feedbackschleife gesehen werden, in der bewertet wird, ob das erstellte Threat Model alle potentiellen Risiken abdeckt oder ob eine weitere Iteration durch die vorherigen Fragestellungen notwendig ist [66].

Während der Durchführung des Threat Modelings ist es relevant, dass Product Owner:innen, Entwickler:innen, Softwarearchitekt:innen und der oder die Security Master:in anwesend sind. Zusätzlich können auch Mitglieder:innen des Quality Assurance Teams involviert werden. Die Erstellung des Modells kann nur im Team erfolgen, da somit eine Vielzahl an unterschiedlichen Meinungen und Sichtweisen vertreten sind. Aus diesem Grund sollte das Team interdisziplinär aufgestellt sein, da somit auch verschiedene Expertisen und Erfahrungen aus unterschiedlichen Bereichen berücksichtigt werden können. Auch unterschiedliche Persönlichkeiten mit verschiedenen Denkanstätzen, wie kritische oder prozessorientierte Denker:innen, können hilfreich sein, um andere Perspektiven auf die Identifikation von möglichen Angriffspunkten zu legen [39].

Die Darstellung des Threat Models kann dabei mittels unterschiedlicher Tools und Praktiken durchgeführt werden. Hierbei ist es vor allem wichtig, dass das Team sich mit der ausgewählten Art wohl fühlt. Beispielsweise können Datenflussdiagramme, Mind-Maps, Sequenzdiagramme, Softwarearchitektur Darstellungen oder auch Listen über Angriffsvektoren und potentielle Security-Probleme eingesetzt werden [39]. Für die Erstellung der verschiedenen Diagramme oder alternativen Darstellungsarten werden häufig bestimmte Verfahren eingesetzt, mit denen Angriffe auf die Systemarchitektur simuliert werden können. Darunter fallen Verfahren wie das von Microsoft entwickelte STRIDE, das P.A.S.T.A. (Process for Attack Simulation and Threat Analysis)-Verfahren oder auch die Anwendung von Top 10 Listen für häufig genutzte Angriffsmöglichkeiten wie die OWASP Top 10 Liste [66].

Im folgenden Abschnitt wird das am häufigsten eingesetzte Verfahren, STRIDE, näher vorgestellt [66]. Das STRIDE-Verfahren hilft dabei Fragen zu sechs spezifischen Kategorien von Risiken zu bilden, anhand derer überprüft werden kann ob das eigene System diese Risiken berücksichtigt. Diese Kategorien umfassen:

- Spoofing: Die Möglichkeit durch den Zugang von anderen User:innen illegalen Zutritt in ein System zu erlangen.
- Tampering: Daten des zu modellierenden Systems unautorisiert zu manipulieren.
- Repudiation: User:innen des Systems können Handlungen durchführen, ohne dass nachvollzogen werden kann, dass diese bestimmten Accounts diese Handlung tatsächlich durchgeführt haben.
- Information Disclosure: User:innen können die Daten von anderen lesen, ohne dass diese absichtlich geteilt wurden.
- Denial of Service: Angreifer:innen könnten das Ziel verfolgen, das System unerreichbar oder unbenutzbar zu machen.
- Elevation of Privilege: Unprivilegierte User:innen können Zugang zu dem gesamten System bekommen, zum Beispiel in Form von Administrator:innen Rechten, wodurch sie die Macht erhalten, das gesamte System zu zerstören.

Diese Kategorien werden nun als Inspiration verwendet, das eigene System zu evaluieren und Lösungsvorschläge zu generieren, um diese Risiken zu verhindern, sofern sie vorhanden sind [20].

Um das Threat Modeling mittels der STRIDE-Methode darzustellen, wurde von Microsoft ein eigenes Tool⁴ entwickelt, mit dem Threat Models erstellt werden können. Das Tool ist frei verfügbar und bietet die Möglichkeit, das Threat Model grafisch darzustellen, siehe Abbildung 4.1. Dazu können vorgefertigte Komponenten per Drag und Drop auf ein Whiteboard gezogen und miteinander verbunden werden. Dadurch lässt sich das eigene System vollständig nachbilden. Anschließend können für jeden Teilbereich des Systems die potentiellen Gefahren festgehalten werden. Dafür werden die entsprechenden STRIDE-Kategorien für jede Komponente vorgeschlagen. Zusätzlich können die identifizierten Gefahren priorisiert werden.

4.2.3 Evil User Stories

Als Alternative oder Ergänzung zur Durchführung und Erstellung eines detaillierten Threat Models können auch sogenannte Evil User Stories eingesetzt werden. Grundsätzlich behandeln User Stories wie sich das System verhält, sofern alles nach Plan läuft, die Umsetzung keine Probleme verursacht und alle User:innen mit guten Absichten mit dem System interagieren. Somit werden Systeme traditionell so konzipiert und weiterentwickelt als ob niemand versuchen würde das System absichtlich anzugreifen. Da dieser Umstand jedoch nicht der Realität entspricht, stellen Evil User Stories, von McGraw auch Abuse Cases genannt, eine Praktik dar, deren Ziel es ist, Hypothesen aufzustellen, wie man ein Softwaresystem oder ausgewählte Teilbereiche davon bestmöglich angreifen kann [27].

⁴<https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool>

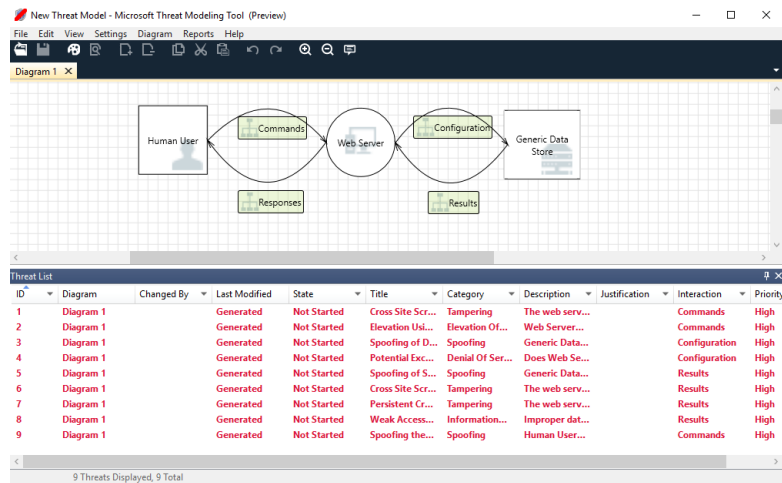


Abbildung 4.1: Microsoft Threat Modeling Tool [46]

Wie auch bei einem Threat Model versetzt man sich bei der Erstellung von Evil User Stories gedanklich in die Rolle der Angreifer:innen und überlegt, wie man am besten in das eigene System eindringen kann. Alternativ kann auch eine Person aus dem Entwicklungsteam damit beauftragt werden, das eigene System kontrolliert anzugreifen, um mit noch einer höheren Wahrscheinlichkeit Schwachstellen ausfindig zu machen. Bei der Überlegung sollte man sich folgende drei Fragen stellen:

- Welche impliziten Annahmen habe ich über das System?
- Wodurch könnten diese Annahmen falsch sein?
- Welche Angriffsmuster könnten Angreifer:innen einsetzen?

Bei der Klärung dieser Fragen ist es vor allem wichtig, dass der oder die Security Master:in des Teams anwesend ist, da es aus Sicht von Entwickler:innen viel schwieriger ist, Schwachstellen im selbst entwickelten System zu identifizieren. Beispielsweise könnte eine implizite Annahme von Entwickler:innen sein, dass User:innen in einem bestimmten Formularfeld nicht mehr als 30 Zeichen eingeben können, da dies über den JavaScript Code entsprechend abgesichert wird. Hierbei besteht jedoch trotzdem die Möglichkeit, dass Angreifer:innen den Request, der zum Server gesendet wird, abfangen und die mitgeschickten Daten manipulieren, wodurch die Anzahl an Zeichen erhöht und es in weiterer Folge zu einem Buffer Overflow kommen könnte. Somit zielt die Entwicklung von Security User Stories darauf ab, durch Fragen aus anderen Perspektiven neue Schwachstelle zu identifizieren [27]. Dabei sollte man sich jedoch bewusst sein, dass Angreifer:innen nicht nur von außerhalb der Organisation, sondern auch aus der eigenen Organisation selbst stammen könnten. Dies könnten beispielsweise unzufriedene oder kürzlich entlassene Mitarbeiter:innen sein. Dadurch wird das Feld der möglichen Angriffsbereiche und Arten nochmals erweitert, da interne Personen andere Zugänge und somit Möglichkeiten für Angriffe haben. Somit ist es relevant, dass man auch dies in den Überlegungen berücksichtigt [56].

Aus den Resultaten lassen sich Evil User Stories formulieren. *Als Angreifer:in möchte ich Daten manipulieren, um einen Buffer Overflow zu erzeugen, damit ich Teile des Speichers überschreiben kann.* Diese Art der User Stories dient jedoch nicht dazu, als Teil des Product Backlogs aufgenommen zu werden, da diese nicht umgesetzt werden sollten. Stattdessen stellen Evil User Stories den ersten Schritt zur Modellierung und Erhebung von potentiellen Bedrohungen dar. Darauf aufbauend können diese Stories als Grundlage für die Erstellung eines ausführlichen grafischen Threat Models herangezogen werden. Alternativ können Evil User Stories auch eigenständig verwendet werden. Dafür würde in einem nächsten Schritt ein weiteres Brainstorming mit einem möglichst interdisziplinär besetzten Team durchgeführt werden bei dem überlegt wird, wie man die gefundene Security Schwachstelle verhindern kann. Anschließend werden Security User Stories erstellt, welche in den Product Backlog aufgenommen und je nach Schweregrad priorisiert und klassifiziert werden, siehe Abschnitt 4.2.4 [56]. Meetings zur Erstellung von neuen Evil User Stories sollten dabei bedarfsabhängig immer dann abgehalten werden, wenn neue Features geplant werden, die in Kürze umgesetzt werden sollten.

4.2.4 Klassifizierung von Security User Stories

Durch das Threat Modeling sowie durch Evil User Stories können verschiedene Arten von Security-Problemen identifiziert werden. Um die daraus resultierenden Security User Stories und Items zu tracken, empfiehlt sich die Einführung eines Projektmanagementsystems, mit dem sich auch die Art und Schwere des Security-Problems übersichtlich erfassen lässt [28]. Hierfür können digitale Lösungen, wie beispielsweise JIRA⁵, aber auch analoge Varianten wie ein Whiteboard mit Post-Its, verwendet werden, um das Scrumban Board und den dazugehörigen Workflow abzubilden. Wichtig ist jedoch, die Möglichkeit, Security User Stories abzubilden sowie diese durch verschiedene Klassifizierungen zu unterscheiden. Die Klassifizierung trägt dazu bei, die Priorität der Bewältigung der Security Schwachstelle sofort für alle Mitglieder:innen des Scrumban Teams ersichtlich zu machen. Außerdem hilft die Priorisierung bei der Einschätzung wie schnell die entdeckte Schwachstelle behoben werden sollte.

Common Vulnerability Scoring System (CVSS)

Bei der Klassifizierung von User Stories spielt die Schwere der gefundenen Security Schwachstelle eine zentrale Rolle. Je nach Schweregrad sollte das Security-Problem entsprechend höher oder niedriger auf dem Scrumban Board priorisiert werden. Eine Möglichkeit, um den Schweregrad von Schwachstellen in der Software abzubilden, stellt der CVSS-Score dar. CVSS setzt dafür einen Algorithmus ein um drei verschiedene Werte zu berechnen: Base-, Temporal- und Environmental-Score. Jeder der Scores kann dabei einen Wert zwischen 0,0 und 10,0 einnehmen, wobei 10,0 für den höchsten Wert und somit die schwerst mögliche Security Schwachstelle steht [18]. Der Base-Score beschreibt die technischen und unveränderlichen Eigenschaften einer Security Schwachstelle. Darauf aufbauend kann der Temporal-Score ermittelt werden, der auch die Veränderung der Schwachstelle über einen Zeitraum hinweg betrachtet. Außerdem kann der Environmental-Score basierend auf dem Base-Score ermittelt werden, der die Umgebung

⁵<https://www.atlassian.com/de/software/jira>

des Systems berücksichtigt. Bei der Berechnung des Base-Scores werden die Voraussetzungen einer Attacke, der Scope des Angriffs sowie die aus einem erfolgreichen Angriff resultierenden Auswirkungen für die Berechnung herangezogen. Im Zuge der Voraussetzungen wird abgefragt, ob die Angreifer:innen physischen oder nur digitalen Zugang zu dem System benötigen, ob ein Angriff auf diese Schwachstelle einen hohen Komplexitätsgrad aufweist, ob angreifende Personen bestimmte Berechtigungen auf dem System benötigen würden und ob Interaktionen mit anderen Benutzer:innen notwendig wären. Für die Berücksichtigung der Konsequenzen wird ermittelt, ob Daten im Zuge des Angriffs gelesen oder auch manipuliert werden konnten. Zusätzlich wird berücksichtigt, ob die Verfügbarkeit des Systems beeinträchtigt wurde. Über den Scope wird erhoben, ob sich die Ausnutzung eines Security-Problems in einem bestimmten Teil des betroffenen Systems auch auf andere Teilbereiche des Systems auswirkt. Auf Basis der Angaben für die entsprechenden Kategorien kann nun der Base-Score errechnet werden. Für die Berechnung sowie für die Angabe der Werte kann der online verfügbare Rechner⁶ des Herausgebers von CVSS, FIRST.Org, verwendet werden. Meistens wird für die Klassifizierung einer Schwachstelle nur der Base-Score verwendet, jedoch sollte vor allem auch der Environmental-Score berücksichtigt werden, da nicht jede Schwachstelle in jedem System einer Organisation einen gleichen Stellenwert hat. Somit empfiehlt es sich, auch die anderen Scores auszufüllen und einen Gesamtscore zu berechnen [59].

Darstellung auf dem Scrumban Board

Das Ergebnis der berechneten Scores kann in fünf qualitative Kategorien für den Schweregrad gegliedert werden: None (0,0), Low (0,1-3,9), Medium (4,0-6,9), High (7,0-8,9) und Critical (8,9-10,0) [59]. Um den berechneten Score für die gefundene Schwachstelle im Projektmanagement-Werkzeug darzustellen, kann ein eigenes Feld in der Ansicht des entsprechenden Arbeitsitems oder der User Story hinzugefügt werden. Hierfür könnte ein Dropdown in der Detailansicht des Items verwendet werden, in dem die qualitativen Kategorien zur Verfügung stehen, um die Zuweisung einfacher zu gestalten. Die ausgewählte Kategorie sollte zur besseren Kennzeichnung ebenfalls auf der Karte, welche auf dem Scrumban Board dargestellt wird, ersichtlich sein. Hierfür könnte eine Farbcodierung, die je nach Kategorie eine andere Farbe verwendet, eingesetzt werden. Alternativ könnten auch Icons fungieren, die die Kategorie repräsentieren, verwendet werden.

Auf Basis der gewählten Klassifizierung der User Story ergibt sich auch die Priorisierung auf dem Scrumban Board. Karten mit der Klassifizierung High oder Critical erfordern die sofortige Umsetzung, um die Ausnutzung von Security Schwachstellen auf der Production-Bereitstellungsumgebung zu verhindern. Um die Priorität dieser Security User Stories auf dem Board noch prominenter darzustellen, sollten die Product Owner:innen diese Karten über allen anderen Karten in der Ready-Spalte platzieren. Zusätzlich kann eine eigene Swimlane auf dem Board eingerichtet werden, in der nur Items platziert werden dürfen, die einen High oder Critical CVSS-Score aufweisen. Zusätzlich sollte den Entwickler:innen vermittelt werden, dass diese Items schnellstmöglich bearbeitet werden sollten. Als Kritikpunkt für den Einsatz des Scores gilt die intransparente Herleitung der Formeln für die Berechnung, da nicht bekannt ist, wie die Kategorien mit ihren jeweiligen Faktoren gewichtet werden. Zusätzlich ist es relevant zu beachten,

⁶<https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator>

dass die Eingabe der Werte für die Berechnung des CVSS-Scores so unvoreingenommen wie möglich geschieht, da ein höherer Wert natürlich mehr Aufmerksamkeit erregt und dazu führt, dass die Behebung der Schwachstelle schneller umgesetzt wird. Somit sollte darauf geachtet werden, dass die Werte für die Kategorien so neutral wie möglich eingegeben werden um zu verhindern, dass jemand durch die Eingabe unrealistischer Werte die Behebung einer grundsätzlich weniger relevanten Schwachstelle durchsetzt [59].

Neben der Schwere der Schwachstelle durch den CVSS-Score sollte auch die Art der Security Vulnerability festgehalten werden, um zu signalisieren, wie die Schwachstelle ausgenutzt werden kann. Zur Klassifizierung der Items können unterschiedliche Kategorien verwendet werden, wobei primär relevant ist, dass jedes Mitglied sich der Bedeutung der Kategorien bewusst ist. Zur Kategorisierung kann das in Abschnitt 4.2.2 vorgestellte STRIDE-Verfahren oder auch die Einträge der OWASP Top 10 Liste herangezogen werden. Alternativ stellt Microsoft im Zuge seines Security Development Lifecycles 15 Kategorien zur Klassifizierung vor. Diese lauten kein Security Bug, Buffer Overflow, arithmetische Fehler, SQL/Skript Injection, Directory Traversal, Race Condition, XSS, Verschlüsselungsfehler, schwache Authentifizierung, unangemessene Berechtigungen, ineffektive Passwortverwaltung, unlimitierter Ressourcenkonsum, inkorrekte oder fehlende Fehlermeldungen, inkorrekte oder fehlende Pfade und eine sonstiges Kategorie [28]. Jeder Security User Story kann nun die entsprechende Kategorie zur Klassifizierung zugeordnet werden.

4.2.5 Security Akzeptanzkriterien

Die zuvor beschriebenen Praktiken fokussieren primär darauf, dass Security Schwachstellen erkannt und in Arbeitsitems dargestellt werden. Ein weiterer relevanter Schritt ist, dass sichergestellt wird, dass die Security Maßnahmen während der Entwicklung umgesetzt werden. Dafür eignet sich die Berücksichtigung von Security bei der Erstellung der Akzeptanzkriterien für die bestimmte User Story. Diese Akzeptanzkriterien dienen zur Kontrolle, dass ein Item nur dann abgeschlossen werden kann, beziehungsweise auf dem Scrumban Board in die Done-Spalte verschoben werden kann, wenn alle Kriterien erfüllt wurden.

Für die Erstellung der Akzeptanzkriterien können die Mitglieder:innen des Scrumban Teams sich an dem erstellten Threat Model sowie an sicheren Coding Praktiken orientieren, welche idealerweise im Rahmen des Security Trainings, siehe Abschnitt 4.1.1, vermittelt wurden. Ein Security Akzeptanzkriterium für ein Feature, welches sich mit der Auswertung von User Input beschäftigt, könnte beispielsweise lauten: *Input Validierungen werden über den Server durchgeführt und alle Validierungsfehler führen zu einem fehlgeschlagenen Request, welcher geloggt wird* [57].

Da sich dieser Abschnitt primär auf die Identifikation und Planung von Security Anforderungen fokussiert, behandelt der folgende Abschnitt Tools, die unterstützend während des Entwicklungs-, Test- und Deploymentsprozesses eingesetzt werden können, um Cyber Security zu berücksichtigen.

4.3 Tools und Praktiken zur Einhaltung von Security Maßnahmen während Entwicklung, Testung und Deployments

Die Einhaltung von Security Maßnahmen während des Entwicklungsprozesses im Rahmen eines Softwareprojektes stellt eine besondere Herausforderung dar. Während der Entwicklung passieren oftmals kleinere Fehler, zum Beispiel das Vergessen eines Semikolons am Ende einer Codezeile. Meistens haben diese Art von Fehlern jedoch keine Auswirkungen, da sie vom Compiler oder der IDE (Integrierte Entwicklungsumgebung) erkannt werden und einen Fehler werfen, welcher von den jeweiligen Entwickler:innen schnell ausgebessert werden kann. Dieser schnelle Feedbackmechanismus lässt sich jedoch auf Fehler in Bezug auf Security nicht anwenden, da hierbei die Entdeckung entweder erst lange nach der Einführung des Fehlers geschieht oder der Fehler nie gefunden wird. Dies führt dazu, dass vulnerabler Programmcode in den Systemen verwendet wird, welcher potentiell von Angreifer:innen ausgenutzt werden kann. Somit empfiehlt sich der Einsatz von Tools zur Analyse von Programmcode auf Security Schwachstellen. Der Vorteil von Tools ist, dass sie im Vergleich von Security Reviews durch Entwickler:innen viel schneller und somit auch kosteneffizienter sind und der Prozess der Security Überprüfungen durch den Einsatz von Tools automatisiert werden kann. Somit besteht kein Mehraufwand für die Entwickler:innen, da sie sich um die Überprüfung nicht manuell kümmern müssen [27]. Zusätzlich können dadurch in kurzer Zeit viele Tests schnell und skalierbar durchgeführt werden und die Organisation kann die zeitlichen Ressourcen der Tester:innen primär auf sehr relevante Teilbereiche und Features des Systems richten [36]. Dabei gilt es zu beachten, dass nur durch den Einsatz von Tools der Softwareentwicklungsprozess nicht automatisch vollständig sicher wird und auch die Überprüfung des Codes durch Entwickler:innen oder Tester:innen nicht vollständig ersetzt werden kann [28]. Dies liegt daran, dass auch Tools nicht alle Fehler finden können und auch manchmal falsche Ergebnisse liefern, in dem sie sicheren Code als Security-Problem identifizieren und umgekehrt. Obwohl viele Tools den Code mit großen Datenbanken von Security-Problemen vergleichen und diese regelmäßig aktualisiert werden, sind die Tools nicht in der Lage sich selbst weiterzuentwickeln und neue Aspekte zu lernen, wie es ein Mensch tun würde [36].

Da Tools dennoch einen großen Mehrwert für die Unterstützung der Sicherheit innerhalb eines agilen Softwareentwicklungsprozesses darstellen, werden in den nachfolgenden Abschnitten Kategorien aus dem Bereich des Testings von Software dargestellt. Diese umfassen Software Composition Analysis (SCA), Static Application Security Testing (SAST) und Dynamic Application Security Testing (DAST). Dabei werden für die unterschiedlichen Kategorien auch Tools vorgestellt, die für diese Kategorie eingesetzt werden können. Im ersten Schritt wird hierbei eine Sammlung an Tools aus den drei Kategorien angelegt. In einem weiteren Schritt werden die gefundenen Tools auf Basis der ISO-Kriterien zur System- und Softwareproduktqualität [42] evaluiert. Zusätzlich werden die Kosten und die Herausgeber:innen der Tools betrachtet. Somit wurden folgende Evaluationskriterien bei der Auswahl berücksichtigt:

- **Funktionalität:** Gibt an, ob das ausgewählte Produkt die gewünschte Funktionalität abdeckt und korrekte Resultate liefert [42].

- Effizienz: Beschäftigt sich mit der Performance und Nutzung der Systemressourcen durch das Produkt [42].
- Kompatibilität: Gibt an, wie gut sich das neue Produkt in die bestehende Infrastruktur eingliedert [42].
- Usability: Beschäftigt sich mit der Einfachheit der Handhabung, der Erlernbarkeit, der Benutzer:innenfreundlichkeit und der Prävention vor Benutzer:innenfehlern des Produkts [42].
- Zuverlässigkeit: Gibt an, ob das ausgewählte Softwareprodukt bei Bedarf verfügbar ist und wie es sich im Falle eines Systemfehlers verhält [42].
- Sicherheit: Gibt an, ob das Produkt Sicherheitsstandards, wie den Schutz der verwendeten Daten, einhält. Zusätzlich wird überprüft, ob das Softwareprodukt auch tatsächlich nur die Funktionalität ausführt, die es angibt durchzuführen.[42].
- Wartung: Gibt an, ob das verwendete Produkt an die gewünschten Anforderungen durch die Anwender:innen angepasst werden kann [42].
- Portabilität: Gibt an, ob die Effektivität des Produktes gemessen oder getestet werden kann. Zusätzlich wird berücksichtigt, ob das Softwareprodukt in anderen Systemen wiederverwendet werden kann [42].
- Kosten: Gibt an, ob es sich bei der gefundenen Software um ein kostenpflichtiges Produkt oder eine kostenlose Open-Source Lösung handelt [58]. Bei den Herausgeber:innen der Tools wird bei Lösungen von Unternehmen sowie bei Open-Source-Produkten die Seriosität der Anbieter:innen überprüft. Hierfür werden die Branche der Herausgeber:innen, sowie wie häufig die entsprechenden Tools von anderen Webseiten empfohlen werden untersucht. Bei Open-Source-Lösungen wird zusätzlich darauf geachtet, wie aktiv an den Lösungen entwickelt wird, um sicherzustellen, dass auch zukünftig Updates für das Tool veröffentlicht werden.
- Herausgeber:innen: Gibt an, ob es sich bei den Herausgeber:innen um eine seriöse und qualitativ hochwertige Quelle handelt [58].

Für die Anforderungen des Unternehmens Tractive wird im Rahmen des Funktionalitätskriterium mittels eines praktischen Tests überprüft, ob das entsprechende Tool die erwarteten Resultate erfüllt. Außerdem sollte das Tool für die von Tractive eingesetzten Packagemanager Gradle, Ruby, Bundler und NPM kompatibel sein. Bei der Effizienz wird ebenfalls anhand eines praktischen Tests überprüft, wie lange die Ausführung der Tools dauert. Hierbei werden Tools mit geringer Ausführungsdauer bevorzugt. Im Zuge der Kompatibilität wird beurteilt wie gut sich das entsprechende Tool in die bestehenden Systeme von Tractive eingliedert. Hierfür wird auf die Kompatibilität mit verwendeten Systemen, wie der Continuous Integration (CI) und Continuous Delivery (CD) Pipeline mit Jenkins⁷, geachtet.

⁷<https://www.jenkins.io/>

4.3.1 Software Composition Analysis

Um in kurzer Zeit große Fortschritte zu erzielen, setzen viele Entwickler:innen auf Frameworks und Libraries von Drittanbietern. Dabei kann es sich um kommerzielle sowie auch um frei verfügbare Open-Source-Lösungen handeln [36]. In beiden Fällen geht es um Quellcode, der in der eigenen Software eingesetzt wird und nicht von den eigenen Entwickler:innen verfasst wurde. Dadurch bindet man jedoch nicht nur die Funktionalität der Drittanbieter-Komponente sondern auch alle Security Schwachstellen, die die jeweilige Komponente enthält, ein. Somit wird auch die eigene Software potentiell unsicherer. Dabei muss es abhängig vom Einsatz und der Struktur der abhängigen Komponente nicht einmal notwendig sein, dass der entsprechende unsichere Code aktiv verwendet wird. Da es in vielen Fällen nicht möglich ist, in der Programmierung auf den Einsatz von externen Libraries und Frameworks zu verzichten, stellt der Einsatz von Software Composition Analysis eine Möglichkeit dar, das Risiko von eingebundenen Security Schwachstellen zu verringern [24].

SCA-Tools basieren auf einer Liste von bekannten vulnerablen Software Komponenten. Diese Liste oder auch Datenbank des jeweiligen Tools kann über öffentlich verfügbare Listen, käuflich erworbene Listen von Unternehmen, Scanning von bestimmten Libraries, Security Expert:innen oder Security Feeds von den Herausgeber:innen der Libraries erstellt werden [24]. Beispielsweise wird die Vulnerability Datenbank der amerikanischen Regierung, National Vulnerability Database (NVD)⁸, häufig von SCA-Tools als Quelle für ihre Liste an Vulnerabilities herangezogen. Vor allem kommerzielle Herausgeber:innen von SCA-Tools setzen jedoch primär auf eigene Listen, in der jedoch oftmals auch die Schwachstellen der NVD eingegliedert werden. Die Liste der Tools wird nun dafür eingesetzt, die Drittanbieter-Abhängigkeiten eines Softwareprojekts mit der Liste abzugleichen und somit zu überprüfen, ob eine oder mehrere der vulnerablen Abhängigkeiten in dem Projekt eingebaut wurden. Um den Abgleich vorzunehmen, analysiert das Tool die Package Manager⁹ der entsprechenden Programmiersprache des Projektes und liest daraus die vorhandenen Abhängigkeiten des Projektes aus. Dabei sollte beachtet werden, ob das Tool bei der Analyse auch jene Abhängigkeiten berücksichtigt, die von den im eigenen Quellcode verwendeten Drittanbieter-Abhängigkeiten verwendet werden, da auch hierbei wiederum Security Schwachstellen vorkommen können [37].

Eine der wichtigsten Faktoren im Bezug auf den Einsatz von Software Composition Analysis ist die Regelmäßigkeit der Durchführung der Scans, da sich der Status der Vulnerabilities in der Datenbank des Tools jederzeit ändern kann und somit auch Komponenten, die kürzlich noch als in Ordnung eingestuft waren, nun eine Sicherheitslücke aufweisen können. Somit sollten am besten täglich oder mindestens einmal in der Woche die entsprechenden Anwendungen gescannt werden, um sicherzustellen, dass neue dem Tool bekannte Schwachstellen sich nicht über einen längeren Zeitraum unentdeckt auf der Production-Bereitstellungsumgebung befinden. Dies gilt auch für Projekte, an denen nur selten gearbeitet wird, da auch hier das gleiche Risiko besteht. Die durchgeführten Scans sollten nach Möglichkeit automatisiert eingesetzt werden, damit keine explizite

⁸<https://nvd.nist.gov/>

⁹Ein Package Manager verwaltet die Abhängigkeiten in einer Programmiersprache. Beispiele für Package Manager sind Gradle (Java, Kotlin), Maven (Java, Kotlin), Bundler (Ruby), Composer (PHP) oder NPM (JavaScript).

Handlungen der Entwickler:innen notwendig ist. Eine Möglichkeit dafür stellt die Integration des Scans in die CI/CD-Pipeline dar. Dafür kann ein eigener Schritt im Rahmen der Pipeline eingebaut werden, welcher automatisch einen SCA-Scan durchführt. Hierbei können Regeln konfiguriert werden, ab wann der Build der Pipeline einen Abbruch erzwingt. Hierbei empfiehlt es sich, den Buildprozess bei Schwachstellen mit potentiell schweren Auswirkungen abubrechen, da ansonsten eine große Sicherheitslücke eingeführt wird. Diese sollte vor einem erfolgreichen Deployment behoben werden.

Eine weitere Möglichkeit zur Einführung von automatisierten SCA-Tests ist zu dem Zeitpunkt, wenn ein neuer Pull Request eröffnet wird. Dadurch kann sofort überprüft werden, ob der neu erzeugte Code Abhängigkeiten mit Security Schwachstellen enthält. Zusätzlich wird dadurch die Frequenz der durchgeführten Tests im Rahmen des Entwicklungsprozesses weiter erhöht. Abschließend sollte berücksichtigt werden, dass durch den Einsatz von SCA-Tools nur bereits bekannte Security-Probleme erkannt werden. Wenn eine Schwachstelle bisher noch nicht gefunden wurde, bleibt diese weiterhin unerkannt. Somit sollten neue Abhängigkeiten mit Bedacht in den eigenen Code übernommen und nur jene verwendet werden, die tatsächlich vollständig benötigt werden [24].

Auf dem Markt für SCA-Tools gibt es eine Vielfalt an unterschiedlichen kommerziellen und kostenfreien Lösungen. Im Bereich der kostenfreien und Open-Source-Lösungen ist zu beobachten, dass die unterstützten Programmiersprachen dieser Tools stark limitiert sind im Vergleich zu kommerziellen Lösungen. Ebenfalls ist der Funktionsumfang dieser Tools vergleichsweise eingeschränkt. Als meistgenannter und häufig empfohlener Vertreter der Open-Source-Kategorie gilt der Dependency-Check¹⁰ von OWASP, welcher sich im Hintergrund auf die NVD Datenbank bezieht. Der Scanner ist mit den Programmiersprachen Java, Kotlin und .NET vollständig kompatibel. Zusätzlich bietet das Tool die Möglichkeit, den Scan über Jenkins in die CI/CD-Pipeline zu integrieren. Als Resultat eines Scans erstellt das Tool einen HTML-Bericht in dem die gefundenen Schwachstellen mit ihrem Schweregrad gekennzeichnet dargestellt werden.

Aus dem Bereich der kommerziellen Anbieter wurde das Tool Snyk Open Source¹¹ aufgrund des großen Funktionsumfangs näher betrachtet. Dieses umfasst die Unterstützung von insgesamt 14 unterschiedlichen Programmiersprachen und 18 Package Managern. Im Funktionsumfang enthalten sind ein Plugin für die JetBrains IDEs, eine CLI, Integrationen mit Bitbucket und JIRA sowie die Integration in die Jenkins Pipeline. Die gefundenen Vulnerabilities werden dabei anhand des CVSS-Scores in der grafischen Ergebnisdarstellung gereiht dargestellt. Zusätzlich werden zu den Ergebnissen auch direkt weiterführende Informationen und Vorschläge zur Behebung der Schwachstellen, durch beispielsweise Versionsupgrades, gegeben. Das Tool speichert ebenfalls die in einem Projekt eingebundenen Drittanbieter-Abhängigkeiten und sendet eine Benachrichtigung sofern sich der Sicherheitsstatus dieser negativ verändert. Weitere Tools neben Snyk Open Source aus dem kommerziellen Bereich sind Debricked, Sonatype, Veracode und WhiteSource.

¹⁰<https://owasp.org/www-project-dependency-check/>

¹¹<https://snyk.io/product/open-source-security-management/>

4.3.2 Static Application Security Testing

Security Schwachstellen können nicht nur durch die Einbindung von vulnerabler Software von Drittanbietern eingeführt werden, sondern auch durch die Entwickler:innen selbst. Eine Möglichkeit, um diese Art von vulnerablen Programmcode mit der Unterstützung eines Tools zu erkennen, ist die Durchführung von Static Application Security Testing (SAST).

Statische Code Analyse analysiert den Code einer Anwendung, ohne dass dieser dabei ausgeführt werden muss [27]. Dabei können Programmcode, Binärdateien oder auch Bytecode auf potentielle Security Schwachstellen überprüft werden. Bei der Evaluierung eines Projektes wird der Programmcode durch das SAST-Tool wie bei einem Compilevorgang geparkt. Jedoch wird anstelle der Verarbeitung des Programmcodes in Binärdatei der Code auf mögliche Security-Probleme untersucht. Dabei kann es sich um die Identifikation von vulnerablen Funktionen und Klassen, unsauber implementierten Security Maßnahmen, ungenutzten Variablen oder auch niedriger Code Qualität handeln [24]. Hierfür wird der Code mittels einer Sammlung an vorab definierten Regeln und Mustern durchsucht. SAST-Tools sollten dabei auch die Möglichkeit zur Definition von eigenen Regeln aufweisen, um das Tool für die eigenen Bedürfnisse zu konfigurieren und somit die Anzahl an gefundenen Schwachstellen zu erhöhen. Dies ist relevant, da, sofern es noch keine Regel für eine bestimmte Art an Security-Problem gibt, das Tool auch keine entsprechenden Schwachstellen finden kann [27]. Zusätzlich generieren SAST-Tools eine Vielzahl an sogenannten False Positives, also Codestücke, die als Security Fehler angezeigt werden obwohl es sich dabei um kein tatsächliches Security-Problem handelt. Dies resultiert daraus, dass man bei der Konfiguration des Tools einstellt, wie sicher das Tool sich bei gefundenen Fehlern sein muss, um diese auch anzuzeigen. Wenn das Tool so konfiguriert wird, dass nur Fehler angezeigt werden, bei denen sich das System nahezu komplett sicher ist, besteht die Gefahr, dass das Tool zwar Security-Probleme vermutet, diese jedoch nicht anzeigt. Dies würde dazu führen, dass eine große Anzahl an Problemen unentdeckt bleibt und stellt somit keine adäquate Konfiguration dar. Somit sollte das Tool so konfiguriert werden, dass es nicht allzu restriktiv vorgeht, obwohl dies zu einer höheren Anzahl an False Positives führt. Aufgrund der potentiell falschen Ergebnisse ersetzt der Einsatz von SAST-Tools nicht die Durchführung von manuellen Security Reviews sondern stellt eine weitere Maßnahme dar, um den Softwareentwicklungsprozess sicherer zu gestalten. Das Ergebnis des SAST-Tool sollte deshalb als Liste von Vorschlägen gesehen werden, welche von Cyber Security geschulten Entwickler:innen mit potentieller Unterstützung von dem oder der Security Master:in evaluiert werden. Um dennoch die bestmöglichen Resultate zu erzielen, ist es relevant, dass das Tool auf die verwendeten Programmiersprachen des jeweiligen Projektes konfiguriert wird [24].

In Bezug auf den Einsatz von SAST-Tools ist die Automatisierung des Scan Prozesses essentiell. Dafür können die Scans bereits sehr früh im Rahmen der Entwicklung eingesetzt werden, da die Software zu dem Zeitpunkt noch nicht ausführbar sein muss. Ein guter Zeitpunkt zur Durchführung des Scans ist im Rahmen des Build Prozesses der Anwendung, da somit jeder neue Code auf Security-Probleme überprüft wird. Zusätzlich bieten manche SAST-Tools auch die Möglichkeit die Scans in Entwicklungsumgebungen oder Code Repositories einzubinden. Somit kann problembehafteter Code bereits erkannt werden während die Entwickler:innen aktiv Code erstellen.

Ebenso wie bei SCA-Tools gibt es eine Vielfalt an unterschiedlichen kommerziellen als auch kostenfreien Lösungen¹². Bei der Auswahl sollte primär darauf geachtet werden, dass das Tool mittels unterschiedlicher Regeln auf die Anforderungen des jeweiligen Projekts konfiguriert werden kann. Ebenfalls sollte die False-Positives-Rate des Tools relativ gering ausfallen, da dies auch die Qualität des Tools repräsentiert. Somit empfiehlt es sich, die entsprechenden Tools zuerst zu testen bevor man sich für den Kauf einer Vollversion entscheidet. Die meisten Tools bieten deshalb auch kostenfreie Demoversionen an. Das am häufigsten empfohlene Tool ist Sonarqube. Dieses lässt sich automatisiert in die CI/CD-Pipeline integrieren und unterstützt 29 verschiedene Programmiersprachen. Zusätzlich lässt sich das Tool auch in Code Repositories wie Github oder Bitbucket einbinden. Sonarqube lässt sich ebenfalls in Entwicklungsumgebungen wie Eclipse, Visual Studio oder die JetBrains Produkte integrieren. Somit kann bereits während der Entwicklung direkt eine erste statische Analyse durchgeführt werden [67].

4.3.3 Dynamic Application Security Testing

Mittels den zuvor beschriebenen, automatisierbaren Testmethoden SAST und SCA wird der Code in einem statischen Zustand überprüft. Diese Testarten können bereits in einer frühen Entwicklungsphase durchgeführt werden, da der Code nicht ausgeführt werden muss. Um jedoch auch Security Schwachstellen zu finden, die nur während der Laufzeit eines Systems identifiziert werden können, sollte zusätzlich auch auf Dynamic Application Security Testing (DAST) gesetzt werden. Damit können Cross-Site-Scripting, SQL-Injections, der allgemeine Umgang mit diversen Arten von Systemeingaben sowie auch allgemeine Security Fehler identifiziert werden. Zusätzlich können durch DAST-Tools fehlerhafte Konfigurationen am Server und auch Probleme mit SSL-Zertifikaten herausgefunden werden [23].

DAST-Tools evaluieren Anwendungen zur Laufzeit des Programms. Dabei wird ein Webbrowser verwendet, um aktiv eine Anwendung anzugreifen und somit Schwachstellen aufzuzeigen. Dadurch sind die durch DAST-Tools gefundenen Schwachstellen nicht nur theoretisch vorhanden, wie bei SAST, sondern bewiesenermaßen existent, wodurch die Rate an falschen Resultaten der Tools erheblich geringer ausfällt. Um die Sicherheitslücken einer Anwendung zu finden, interagieren DAST-Scanner mit einer Anwendung, wie es auch menschliche Nutzer:innen tun würden. Anfänglich versuchen die Scanner die Architektur der jeweiligen Anwendung zu verstehen, indem sie die verschiedenen Bereiche der Anwendung mit einem Crawler auslesen. Dabei können Formulare, Links, Texte sowie weitere Elemente der Anwendung, mit denen User:innen interagieren könnten, identifiziert werden. Zusätzlich werden auch potentielle Angriffspunkte, wie Werte im Header, URL-Parameter und Cookies erfasst. Nach Abschluss des Crawlings überprüft das DAST-Tool alle gefundenen Objekte und greift diese automatisiert an [23]. Dafür wird von den DAST-Tools Fuzzing eingesetzt. Dies ist eine Technik des Software Testings, bei der ungültige, zufällige und unerwartete Eingabedaten in der Anwendung angewandt werden. Darunter fallen Eingaben mit mehreren tausend Zeichen, Shell Codestücke, Script HTML-Tags mit JavaScript Code und auch SQL-Abfragestrings. An-

¹²Eine Liste von möglichen von OWASP zusammengestellten Tools findet sich unter <https://tinyurl.com/y234bxph> gefunden werden.

schließlich wird das attackierte Programm auf daraus resultierende Fehler, Abstürze oder Datenleaks überwacht. Der Testprozess wird dabei vollständig automatisiert durchgeführt, wodurch die Tools einen großen Vorteil gegenüber von Menschen durchgeführten Tests haben, da hierbei in kurzer Zeit mehrere Aufgaben gleichzeitig und ausführlicher durchgeführt werden. Nichtsdestotrotz wird für die Auswertung der Ergebnisse eine Person mit Security Kenntnissen, beispielsweise der oder die Security Master:in, benötigt, um potentiell gefundene Schwachstellen zu verifizieren und entsprechend priorisiert in den Backlog einzufügen [24]. Dabei sollte auch mit den Entwickler:innen Rücksprache gehalten werden, da DAST-Tools zwar Schwachstellen ausfindig machen, jedoch aufgrund der Tatsache, dass der Quellcode nicht analysiert wird, es keine Hinweise darauf gibt, an welcher Stelle im Code das Problem verursacht wird [62].

Wie auch bei den anderen Arten von Security Testing ist beim Einsatz von DAST vor allem darauf zu achten, dass der Prozess automatisiert im Scrumban Entwicklungsprozess durchgeführt werden kann. Dies sollte jedoch nicht auf der Production-Bereitstellungsumgebung durchgeführt werden, um potentielle Komplikationen, die durch das aktive Vorgehen während dem Scanning verursacht werden, zu vermeiden.

Ein weiterer Aspekt der beim Einsatz von DAST berücksichtigt werden sollte, ist, dass die Ausführung der Scans je nach Größe der zu scannenden Applikation mehrere Stunden bis Tage dauern kann. Somit würde ein vollständig durchgeführter Scan im Zuge eines automatisierten Deployments in Rahmen der CI/CD-Pipeline potentiell zu lange dauern. Deshalb bieten die DAST-Tools die Möglichkeit den zu scannenden Bereich der Anwendung einzugrenzen. Alternativ könnte der DAST-Scan auch separat außerhalb von Feature Deployments automatisiert durchgeführt werden, wodurch schnell benötigte Deployments nicht über einen längeren Zeitraum aufgehalten werden. Zusätzlich ist für eine optimale Konfiguration des Tools sowie für die Evaluation der Ergebnisse der oder die Security Master:in notwendig, um effektive Testeinstellungen zu konfigurieren [62].

Auch für DAST gibt es verschiedenste Lösungen. Bei der Auswahl gilt es jedoch darauf zu achten, dass das Tool auch automatisiert eingesetzt werden kann, da es auch einige Tools gibt, die nur manuelle Scans ermöglichen. Kostenpflichtige Lösungen sind beispielsweise Acunetix, Forify WebInspect, Netsparker oder Appknox. Alternativ können auch kostenlose Lösungen wie OWASP ZAP eingesetzt werden.

4.3.4 Security Unit Testing

Eine weitere Möglichkeit um sicherzustellen, dass der geschriebene Programmcode Security Standards erfüllt, ist die Implementierung von Security Unit Tests. Diese können dafür eingesetzt werden, die im Rahmen der Security User Stories definierten Akzeptanzkriterien programmatisch festzuhalten und deren Einhaltung zu überprüfen.

Unit Tests testen jeweils nur einen kleinen Ausschnitt einer Software. Das Ziel davon ist sicherzustellen, dass jede kleine Einheit des gesamten Programms sich so verhält wie gewünscht. Ein einzelner Unit Test ist für die kleinst möglich testbare Einheit zuständig und hat dabei normalerweise nur ein paar wenige Input Parameter und einen einzigen Output. Unit Tests werden dabei häufig im gleichen Code Repository wie das eigentliche Programm gespeichert. Idealerweise werden alle Unit Tests durchlaufen sobald Entwickler:innen neue Software in die Versionsverwaltung einchecken. Falls diese

nicht erfolgreich absolviert werden, müssen die Entwickler:innen den Code anpassen. Grundsätzlich werden Unit Tests positiv geschrieben, wodurch man überprüft, ob der Code das Ergebnis erzeugt welches er erzeugen sollte. Dieses Verhalten lässt sich jedoch auch umdrehen, um zu verifizieren, ob das Software Teilstück auch mit negativem Input adäquat umgeht. Somit können auch Security Anforderungen überprüft werden. Die für das entsprechende Code Stück notwendigen Tests in Bezug auf die Security können den Security Akzeptanzkriterien, dem Threat Model oder auch bestehenden Listen von häufigen Security Bedrohungen, wie den OWASP Top 10, entnommen werden [24].

Ein beispielhaftes Testszenario für einen Unit Test für eine Funktion, welche die von User:innen eingegebenen Werte eines Input Feldes in der Datenbank speichert, wäre zu überprüfen, was passiert, wenn anstelle einer regulären Eingabe ein SQL-Statement als String an die Funktion übergeben wird. Somit könnte der Unit Test als Parameter folgenden String verwenden: „Test; DROP TABLE Users;“. Die Aufgabe des Tests ist es somit zu validieren, dass dieses Statement nicht als SQL-Befehl auf der Datenbank ausgeführt wird.

Sicherheit kann im Rahmen der Softwareentwicklung jedoch nicht nur durch die Einführung von Tools und Testmethoden gewährleistet werden. Darum geht der nächste Abschnitt auf Events ein, welche die Einhaltung von Cyber Security Maßnahmen im Rahmen des Scrumban Prozesses unterstützen.

4.4 Secure Scrumban

Neben der Einführung von Tools zur Überprüfung des Programmcodes auf Security Schwachstellen sollte der Programmcode weiterhin auch durch Menschen überprüft werden. Dies ist notwendig, um die Ergebnisse der Tools zu interpretieren aber auch um komplexe Softwarearchitekturen und Umsetzungen zu evaluieren [27]. Für die Umsetzung der Reviews werden eigene Events und Meetings in den folgenden Abschnitten empfohlen. Zusätzlich gibt dieser Abschnitt einen Überblick über den vorgeschlagenen Secure Scrumban Prozess.

4.4.1 Secure Code Review

Im Rahmen eines Secure Code Reviews wird Programmcode manuell auf Probleme in Bezug auf Cyber Security überprüft. Im Zuge des Reviews ist es nicht notwendig, dass alle Zeilen eines Features vollständig evaluiert werden. Primär werden nur jene Programmstellen untersucht, die eine Security Funktion erfüllen. Zusätzlich analysiert man jene Stellen im Code, bei denen verschiedene Systemteile ineinander greifen. Dabei kann es sich beispielsweise um ein Frontend handeln, welches einen API-Aufruf durchführt oder auch um eine Anwendung, die Fehler in eine Log-Datei speichert, welche sich auf einem anderen Server befindet. Um die entsprechenden Stellen im Programmcode zu identifizieren, können die Security Akzeptanzkriterien sowie auch das für die jeweilige Anwendung erstellte Threat Model herangezogen werden. Während des Review Prozesses wird darauf geachtet, dass bei den untersuchten Stellen für unterschiedliche Szenarien Vorkehrungen getroffen wurden. Diese Szenarien sollten dabei unter anderem Authentifizierung, Autorisierung, Security Header, Verschlüsselung, Identitätsmanagement, Input

Validierung und Session Management berücksichtigen. Zusätzlich sollte auch auf Spezifika der eingesetzten Programmiersprachen, wie unsichere Funktionsaufrufe, eingegangen werden. Um den Prozess zu vereinfachen, gibt es bereits vordefinierte Checklisten¹³, die online auffindbar sind und bei der Durchführung des Reviews unterstützend eingesetzt werden können [24].

Ein Secure Code Review kann in unterschiedlichen Phasen des Scrumban Entwicklungsprozesses durchgeführt werden. Dies kann bereits während der aktiven Programmierung des entsprechenden Arbeitsitems, zu dem Zeitpunkt bevor der neue Code merged wird oder auch erst kurz bevor dieser deployed wird, stattfinden. Dabei werden jedoch ausschließlich nur die durch den neu entwickelten Code betroffenen Stellen in der Anwendung betrachtet und nicht der gesamte Code, da ansonsten der gleiche Code bei jedem Review analysiert wird und die Dauer des Reviews stark ansteigt. Bei der Durchführung des Reviews ist es empfehlenswert, andere Entwickler:innen oder auch den oder die Security Master:in zu bitten, das Review gemeinsam durchzuführen, um die Wahrscheinlichkeit von unentdeckten Security-Problemen zu minimieren [24].

4.4.2 Security Retrospektive

Die Security Retrospektive beschreibt ein Event bei dem die Security-relevanten Rollen, Verantwortlichkeiten und Prozesse evaluiert werden [16]. Diese Retrospektive sollte dabei mindestens in einem jährlichen Zyklus durchgeführt werden. Jedoch ist wichtig zeitnah auf aktuelle Entwicklungen einzugehen, da sich die Anforderungen in agilen Prozessen kontinuierlich verändern können. Dabei kann es sich um neue eingesetzte Technologien, Personalveränderungen, Veränderungen in der Unternehmensorganisation, ein Cyber-Security-Vorfall, neue Cyber-Security-Bedrohungen oder auch gesetzliche Änderungen handeln. Während der Retrospektive sollten sich die Teilnehmer:innen folgende Fragen stellen:

- Sind die aktuellen Prozesse noch zeitgemäß?
- Wie einfach ist es, diesen Prozessen und Vorgehensweisen zu folgen?
- Wurden in letzter Zeit neue Technologien eingeführt, die noch nicht über die Security Maßnahmen abgedeckt werden?
- Sind die Mitarbeiter:innen für die Umsetzung der Maßnahmen ausreichend ausgebildet oder ist weiteres Security Training notwendig?

Dabei ist es wichtig, die Meinungen aller Stakeholder:innen (Entwickler:innen, Product Manager:innen, Security Master:innen und Führungspersonen) zu berücksichtigen, um gegebenenfalls die aktuellen Prozesse und Maßnahmen zu adaptieren [61].

4.4.3 Secure Scrumban Prozess

Abbildung 4.2 ordnet die in den vorherigen Kapiteln vorgestellten Maßnahmen, in roter Farbe, in den Scrumban Prozess ein und zeigt somit auf, an welchen Stellen Security

¹³ Beispielsweise <https://snyk.io/wp-content/uploads/Snyk-Secure-Code-Review-Cheat-Sheet.pdf>

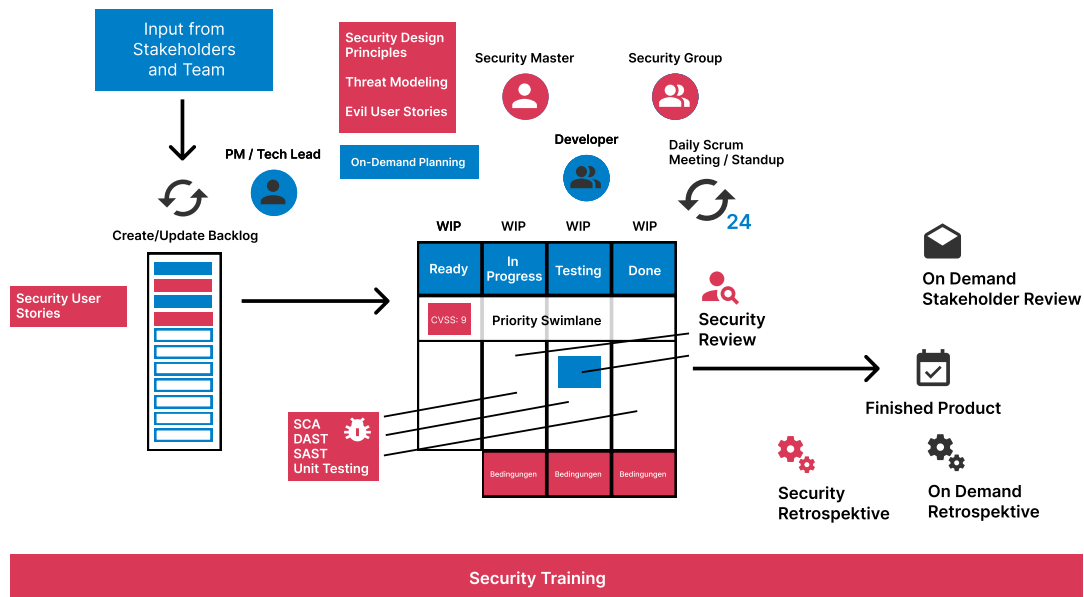


Abbildung 4.2: Secure Scrumban Prozess

Maßnahmen inkludiert werden können. Der im Rahmen dieser Arbeit vorgeschlagene Prozess zur Integration von Cyber-Security-Maßnahmen berücksichtigt Maßnahmen die den gesamten Prozess von der Planung bis zum finalisierten Produkt sowie dessen Weiterentwicklungen begleiten. Dies ist relevant, da in jedem Schritt die Möglichkeit besteht, dass potentielle Schwachstellen eingeführt oder nicht berücksichtigt werden. Durch die Einhaltung der Maßnahmen wird dieses Risiko minimiert.

Kapitel 5

Fazit

Unter Cyber Security versteht man jegliche Maßnahmen, die dazu beitragen, den Cyberspace vor unrechtmäßiger Beanspruchung, welche auf unterschiedliche Arten geschehen kann, zu schützen. Da die Planung von entsprechenden Schutzmaßnahmen und Security Anforderungen in agilen Prozessmodellen aufgrund der sich kontinuierlich ändernden Anforderungen nicht vollständig zu Beginn erfolgen kann, ist es relevant, Sicherheitsmaßnahmen laufend in allen Phasen des Entwicklungsprozesses einzubinden und kontinuierlich anzupassen. Auf Basis einer Analyse bestehender Secure Software Development Modelle wurden Best Practices und Tools für einen sicheren Scrumban Prozess abgeleitet. Einen wichtigen Faktor stellt dabei das Training der Mitglieder:innen des Scrumban Teams dar. Hierfür eignet sich der Einsatz von auf Rollen und Technologien zugeschnittenen Trainings. Dabei ist ein regelmäßiges, kontinuierliches Lernen mit aktuellen Inhalten von Bedeutung. Das Scrumban Team sollte zusätzlich um Security-spezifische Rollen, wie den oder die Security Master:in und einer Security Gruppe ergänzt werden, um Unterstützung bei Security Maßnahmen und Fragen zu erhalten.

Allgemein sollte auf das Pushing-Left-Prinzip gesetzt werden, wodurch Security frühestmöglich im Prozess berücksichtigt wird. Somit sollten nicht-funktionale Anforderungen wie Security auch im Product Backlog bereits inkludiert werden. Diese Anforderungen können dabei durch die Erstellung eines Threat Models, Evil User Stories sowie der Identifikation von häufigen Risiken für das Software Design erhoben werden. Um die Anforderungen in den Backlog zu überführen, sollten Security User Stories erstellt werden. Um die Wichtigkeit der Umsetzung und somit die Priorisierung dieser Stories vorzunehmen, kann der CVSS-Score eingesetzt werden. Stories mit einem Score aus den Kategorien High oder Critical erfordern dabei eine schnelle Umsetzung, um diese Schwachstellen zu schließen. Dafür sollten die entsprechenden Karten auf dem Scrumban Board gekennzeichnet werden und in eine separate Swimlane mit hoher Priorität gezogen werden. Um zu kontrollieren, ob eines der Arbeitsitems auf dem Board hinsichtlich der Security Anforderungen abgeschlossen werden kann, können Security Akzeptanzkriterien für jedes Item definiert werden. Diese müssen abgeschlossen werden, bevor das Item auf dem Board verschoben werden darf.

Um den Einsatz von Security Maßnahmen während des Entwicklungsprozesses zu fördern und zu kontrollieren, empfiehlt es sich, auf automatisierbare Tools zur Analyse des Codes und der Anwendung zurückzugreifen. Dabei können SCA, SAST, DAST sowie auch Unit Tests eingesetzt werden. Obwohl Tools eine wichtige Komponente zur

Überprüfung von Security Schwachstellen sind, ersetzen sie nicht das von Menschen durchgeführte Security Review. Dieses kann an beliebigen Zeitpunkten im Prozess, idealerweise gemeinsam mit anderen Entwickler:innen, durchgeführt werden. Zusätzlich ist es relevant, die im Scrumban Prozess eingesetzten Security Maßnahmen, Prozesse und Rollen nach Bedarf anzupassen, um auf technische sowie menschliche Veränderungen reagieren zu können.

Das Thema der Cyber Security in agilen Prozessmodellen kann aufgrund seiner Tiefe und Komplexität nur zu einem kleinen Teil in dieser Arbeit behandelt werden. Daher wurde das Thema der Arbeit auf das Prozessmodell Scrumban eingeschränkt und fokussiert darauf, wie der Scrumban Ablauf durch die Einführung von Security Maßnahmen sicherer gestaltet werden kann. Die dabei vorgestellten Maßnahmen, Praktiken und Tools wurden auf Basis einer Literaturrecherche und bestehenden Secure Software Development Lifecycles ausgewählt. Allgemein sollte bei dem vorgestellten Secure Scrumban Prozess darauf geachtet werden, dass es sich dabei um ein Modell handelt, welches stark geordnete Abläufe vorgibt, die in der Realität meist chaotischer und weniger stark gegliedert umgesetzt werden [12]. Im Zuge der Einführung des Prozesses ist es sinnvoll, dass dies schrittweise geschieht und nicht alle Maßnahmen auf einmal eingeführt werden. Somit kann verhindert werden, dass die Praktiken als Belastung für die existierenden Scrumban Teams der jeweiligen Organisation empfunden werden. Eine schrittweise Einführung bietet außerdem den Vorteil, dass die Teammitglieder:innen Feedback zu den Maßnahmen geben und somit direkt Adaptionen vorgenommen werden können, um eine bessere Passung zwischen den Praktiken des vorgeschlagenen Secure Scrumban Prozesses sowie den bestehenden Arbeitsweisen des Teams herzustellen.

Bei dem vorgestellten Secure Scrumban Prozess gilt es ebenfalls zu beachten, dass es selbst durch dessen Einführung und einer vollständigen Umsetzung aller vorgeschlagenen Praktiken und Tools keine Garantie dafür gibt, dass es nie zu einem Security Vorfall kommen wird. Letztlich ist kein Mensch und kein Tool in der Lage, alle Szenarien zu evaluieren, die zu einer Sicherheitslücke führen können. Davon abgesehen entwickeln sich auch die Angreifer:innen immer weiter und versuchen den Tools und bestehenden Sicherheitsmaßnahmen einen Schritt voraus zu sein, um Anwendungen zu attackieren. Aus diesem Grund ist es ebenfalls wichtig zu beachten, dass die in dieser Arbeit beschriebenen Praktiken nur eine Momentaufnahme der bestehenden Empfehlungen zur Sicherung des agilen Softwareentwicklungsprozesses darstellen. Aufgrund der sich kontinuierlich weiterentwickelnden Angriffsmethoden und Vorgehensweisen ist es wichtig, dass die Prozesse und Maßnahmen regelmäßig bei Bedarf aktualisiert und ausgetauscht werden, um mit den aktuellen Entwicklungen mithalten zu können.

Abbildungsverzeichnis

2.1	Schritte des iterativ-inkrementellen Vorgehens [45]	4
2.2	Scrum Prozess [48]	6
2.3	Beispiel eines Kanban Boards [1]	8
2.4	Beispiel eines Scrumban Boards [47]	10
2.5	Darstellung eines Scrumban Prozesses	11
4.1	Microsoft Threat Modeling Tool [46]	29
4.2	Secure Scrumban Prozess	42

Tabellenverzeichnis

3.1	Analyse bestehender Security Modelle	18
-----	------------------------------------------------	----

Quellenverzeichnis

- [1] David Anderson und Andy Carmichael. *Die Essenz von Kanban kompakt*. 1. Aufl. Heidelberg, DE: dpunkt.verlag, 2017.
- [2] Zulkarnain Azham, Imran Ghani und Nora Ithnin. „Integrating Software Security into Agile-Scrum Method“. *KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS* 8 (Feb. 2014).
- [3] Abdelelah Babiker, Adil Mahmoud und Alameen Abdalrahman. „Sprint Backlog Estimating and Planning Using Planning Poker Technique in Agile Scrum Framework“. *International Journal of Advanced Research in Computer Science and Software Engineering* 8 (Mai 2018).
- [4] Victor R. Basili und Albert J. Turner. „Iterative enhancement: A practical technique for software development“. *IEEE Transactions on Software Engineering* SE-1.4 (1975), S. 390–396.
- [5] Kelly Bissell, Ryan M. LaSalle und Paolo Dal Cin. *Ninth Annual Cost of Cybercrime Study*. Techn. Ber. 9. Dublin, IRL: Accenture, März 2019. URL: https://www.accenture.com/_acnmedia/PDF-96/Accenture-2019-Cost-of-Cybercrime-Study-Final.pdf.
- [6] Kelly Bissell u. a. *State of Cybersecurity Resilience 2021*. Techn. Ber. 165. Dublin, IRL: Accenture, Nov. 2021. URL: https://www.accenture.com/_acnmedia/PDF-165/Accenture-State-Of-Cybersecurity-2021.pdf.
- [7] BSIMM. *BSIMM12*. Techn. Ber. 12. USA: BSIMM - Building Security In Maturity Model, Dez. 2021. URL: <https://www.bsimm.com/content/dam/bsimm/reports/bsimm12.pdf?elqTrackId=88dcbfc942e24ed6902ed828a921f601&elq=7a85ecb31f77495d9717a050bb59c6cb&elqaid=2821&elqat=1&elqCampaignId=&elqcst=272&elqcside=1538>.
- [8] Brian Chess und Jacob West. *Secure Programming with Static Analysis*. 1. Aufl. Boston, MA: Addison-Wesley Professional, 2007.
- [9] David Cohen, Maikael Lindvall und Patricia Costa. „An Introduction to Agile Methods“. In: *Advances in Computers - Advances in Software Engineering*. Hrsg. von Marvin Zelkowitz. San Diego, CA: Elsevier Academic Press, 2004. Kap. 1, S. 2–63.
- [10] Harris M. Cooper. *Integrating Research: A Guide for Literature Reviews (Applied Social Research Methods)*. 2. Aufl. Thousand Oaks, CA: Sage Publications, Inc., 1989.

- [11] Dan Craigen, Nadia Diakun-Thibault und Randy Purse. „Defining Cybersecurity“. *Technology Innovation Management Review* 4 (Okt. 2014).
- [12] Thomas G. Cummings und Christopher G. Worley. *Organization Development and Change*. 1. Aufl. Stamford, USA: Cengage Learning, 2019.
- [13] Noopur Davis. *Secure Software Development Life Cycle Processes: A Technology Scouting Report*. Techn. Ber. CMU/SEI-2005-TN-024. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, Dez. 2012. URL: <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7457>.
- [14] Kelley Dempsey u. a. *Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations*. Techn. Ber. 800-137. USA: NIST - National Institute of Standards und Technology, Sep. 2011. URL: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-137.pdf>.
- [15] Digital.ai. *15th Annual State Of Agile Report*. Techn. Ber. 15. USA: Digital.ai, Apr. 2021. URL: <https://itnove.com/wp-content/uploads/2021/07/15th-state-of-agile-report.pdf>.
- [16] Donna Dodson, Murugiah Souppaya und Karen Scarfone. *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*. Techn. Ber. 1.1. USA: NIST - National Institute of Standards und Technology U.S. Departement of Commerce, Feb. 2022. URL: <https://doi.org/10.6028/NIST.SP.800-218>.
- [17] Dusten James Doggett. „Removing Buffer Overflows In C Programs With Safe Library Replacement Transformation“. Magisterarb. Auburn: Auburn University, Dez. 2013.
- [18] FIRST.Org. *Common Vulnerability Scoring System version 3.1*. Techn. Ber. 1. USA: FIRST.Org, Juni 2019. URL: https://www.first.org/cvss/v3-1/cvss-v31-user-guide_r1.pdf.
- [19] Lynn Ann Fitcher und von Solmsm Roussouw. „SecSDM: A Model for Integrating Security into the Software Development Life Cycle“. In: *International Federation for Information Processing Digital Library; Fifth World Conference on Information Security Education*; (New York, NY). Hrsg. von Lynn Fitcher und Ronald Dodge. New York, NY: Springer US, Jan. 2007, S. 41–48.
- [20] Jeremy Geib u. a. *Microsoft Threat Modeling Tool threats*. Techn. Ber. 1. Redmond, WA: Microsoft Corporation, März 2022. URL: <https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats>.
- [21] The PHP Group. *exec*. 1. Jan. 2022. URL: <https://www.php.net/manual/de/function.exec.php>.
- [22] Mareike Hattendorf. „Die Aktualität des Agilen - Qualitative Forschung im Bereich digitaler Dienstleistungsunternehmen“. *Human Resource Development Review* 1.29 (2021), S. 93–119.
- [23] Veracode Inc. *Veracode Dynamic Analysis*. Techn. Ber. 1. Burlington, MA: Veracode Inc., Jan. 2020. URL: <https://www.veracode.com/sites/default/files/pdf/resources/whitepapers/dynamic-analysis-white-paper-veracode.pdf>.

- [24] Tanya Janca. *Alice and Bob learn Application Security*. 1. Aufl. Indianapolis, IN: Wiley, 2021.
- [25] Muhammad Umair Ahmed Khan und Mohammed Zulkernine. „On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software“. In: *33rd Annual IEEE International Computer Software and Applications Conference* (Seattle, WA). Hrsg. von Tony Hey. Los Alamitos, CA: IEEE Computer Society, Aug. 2009, S. 353–358.
- [26] Corey Ladas. *Scrumban - And Other Essays on Kanban Systems for Lean Software Development*. 1. Aufl. Seattle, WA: Modus Cooperandi Press, 2008.
- [27] Gary McGraw. *Software Security - Building Security In*. 1. Aufl. Boston, MA: Addison-Wesley Professional, 2006.
- [28] Microsoft. *Security Development Lifecycle - SDL Process Guidance Version 5.2*. Techn. Ber. 5.2. Redmond, WA: Microsoft Corporation, Mai 2012. URL: <https://www.microsoft.com/en-us/download/details.aspx?id=1237>.
- [29] Juan de Vicente Mohino u. a. „The Application of a New Secure Software Development Life Cycle (S-SDLC) with Agile Methodologies“. *Electronics* 8 (Okt. 2019).
- [30] OWASP. *OWASP SAMM*. Techn. Ber. 2. Mountain View, CA: Open Web Application Security Project, März 2021. URL: <https://owaspsamm.org/model/>.
- [31] Ed Paradise. *Cisco Secure Development Lifecycle*. Techn. Ber. 1. San Jose, CA: Cisco Systems Inc., Jan. 2021. URL: https://www.cisco.com/c/dam/en_us/about/doing_business/trust-center/docs/cisco-secure-development-lifecycle.pdf.
- [32] Balasubramaniam Ramesh, Lan Cao und Richard Baskerville. „Agile requirements engineering practices and challenges: An empirical study“. *Information Systems Journal* 20 (Sep. 2010).
- [33] Ajay Reddy. *The Scrumban [R]Evolution - Getting the Most Out of Agile, Scrum, and Lean Kanban*. 1. Aufl. Boston, MA: Addison-Wesley Professional, 2016.
- [34] Kalle Rindell, Sami Hyrynsalmi und Ville Leppänen. „Busting a Myth: Review of Agile Security Engineering Methods“. In: *ARES '17: Proceedings of the 12th International Conference on Availability, Reliability and Security* (Reggio Calabria, Italy). Hrsg. von Edgar Weippl und A Min Tjoa. New York, NY, USA: Association for Computing Machinery, Aug. 2017, S. 1–10.
- [35] Kalle Rindell u. a. „Security in agile software development: A practitioner survey“. *Information and Software Technology* 131 (März 2021).
- [36] SAFECODE. *Fundamental Practices for Secure Software Development*. Techn. Ber. 3. USA: Software Assurance Forum for Excellence in Code, März 2018. URL: http://safecode.org/wp-content/uploads/2018/03/SAFECODE_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf.
- [37] SAFECODE. *Managing Security Risks Inherent in the Use of Thirdparty Components*. Techn. Ber. 1. USA: Software Assurance Forum for Excellence in Code, Jan. 2017. URL: https://safecode.org/wp-content/uploads/2017/05/SAFECODE_TPC_Whitepaper.pdf.

- [38] SAFECODE. *Practical Security Stories and Security Tasks for Agile Development Environments*. Techn. Ber. 1. USA: Software Assurance Forum for Excellence in Code, Juli 2012. URL: https://safecode.org/wp-content/uploads/2018/01/SAFECode_Agile_Dev_Security0712.pdf.
- [39] SAFECODE. *Tactical Threat Modeling*. Techn. Ber. 1. USA: Software Assurance Forum for Excellence in Code, Jan. 2017. URL: https://safecode.org/wp-content/uploads/2017/05/SAFECode_TM_Whitepaper.pdf.
- [40] Jerome H. Saltzer und Michael D. Schroeder. „The protection of information in computer systems“. *Proceedings of the IEEE* 63 (Sep. 1975).
- [41] Ken Schwaber und Jeff Sutherland. *The Scrum Guide*. 1. Nov. 2020. URL: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>.
- [42] ISO Central Secretary. *ISO/IEC 25010:2011 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. Techn. Ber. ISO/IEC 25010:2011. CH: International Organization for Standardization, März 2011. URL: <https://www.iso.org/standard/35733.html>.
- [43] Marianne Swanson u. a. *Contingency Planning Guide for Federal Information Systems*. Techn. Ber. 800-34. USA: NIST - National Institute of Standards und Technology, Mai 2010. URL: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-34r1.pdf>.
- [44] Richard J. Torraco. „Writing Integrative Literature Reviews: Guidelines and Examples“. *Human Resource Development Review* 4.3 (2005), S. 356–367.
- [45] Achilleas Anagnostopoulos. *The steps of the interactive enhancement model*. Jan. 2020. URL: <https://www.oreilly.com/library/view/hands-on-software-engineering/9781838554491/assets/b3033f98-8586-4b11-baa2-128ed0225b0d.png>.
- [46] Jeremy Geib u. a. *Day-in-a-Life of a Scrum Team*. Sep. 2020. URL: <https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-feature-overview>.
- [47] Corey Ladas. *Backlog Limit*. Jan. 2008. URL: <https://www.agilealliance.org/scrumban/>.
- [48] Scrum.org. *The Scrum Framework Poster*. Jan. 2022. URL: <https://www.scrum.org/resources/scrum-framework-poster>.
- [49] Accenture. *Triple digit increase in cyberattacks: What next?* 4. Aug. 2021. URL: <https://www.accenture.com/us-en/blogs/security/triple-digit-increase-cyberattacks> (besucht am 12. 12. 2021).
- [50] Agile Alliance. *Agile 101*. 1. Jan. 2022. URL: <https://www.agilealliance.org/agile101/> (besucht am 07. 03. 2022).
- [51] Kent Beck u. a. *Manifesto for Agile Software Development*. 13. Feb. 2001. URL: <http://www.agilemanifesto.org/> (besucht am 24. 02. 2022).
- [52] Zoe Braiterman u. a. *Threat Modeling Manifesto*. 29. Aug. 2020. URL: <https://www.threatmodelingmanifesto.org/> (besucht am 04. 04. 2022).

- [53] Eric Browning. *Security and agile: How competing goals can meet in the middle*. 17. März 2017. URL: <https://techbeacon.com/app-dev-testing/security-agile-how-competing-goals-can-meet-middle> (besucht am 15.03.2022).
- [54] Digite. *What is Kanban - Overview of the Kanban Method*. 1. Jan. 2022. URL: <https://www.digite.com/kanban/what-is-kanban> (besucht am 10.03.2022).
- [55] Jim Highsmith. *History: The Agile Manifesto*. 13. Feb. 2001. URL: <https://agilemanifesto.org/history.html> (besucht am 24.02.2022).
- [56] Jeremy Jarrell. *Embracing Evil with Evil User Stories*. 19. Okt. 2017. URL: <https://www.pivotaltracker.com/blog/embracing-evil-user-stories> (besucht am 13.04.2022).
- [57] Eric Johnson. *Security Acceptance Criteria*. 20. Aug. 2018. URL: <https://github.com/OWASP/user-security-stories/blob/master/security-acceptance-criteria.md> (besucht am 12.04.2022).
- [58] Mathias Knops. *Software Selection Process and Criteria*. 1. Jan. 2020. URL: <https://blog.software-advisory.com/software-selection-process-and-criteria> (besucht am 06.02.2022).
- [59] Andreas Kurtz. *Von niedrig bis kritisch: Schwachstellenbewertung mit CVSS*. 25. Jan. 2021. URL: <https://www.heise.de/hintergrund/Von-niedrig-bis-kritisch-Schwachstellenbewertung-mit-CVSS-5031983.html> (besucht am 11.04.2022).
- [60] Developer Mozilla. *eval()*. 25. Jan. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval (besucht am 27.03.2022).
- [61] Anirudh Nadkarni. *How Often Should You Review Your Policies and Procedures?* 14. Mai 2019. URL: <https://blog.24by7security.com/how-often-should-you-review-your-policies-and-procedures> (besucht am 26.04.2022).
- [62] Julie Peterson. *Dynamic Application Security Testing: DAST Basics*. 30. Apr. 2021. URL: <https://www.whitesourcesoftware.com/resources/blog/dast-dynamic-application-security-testing/> (besucht am 24.04.2022).
- [63] Open Web Application Security Project. *Fuzzing*. 27. Juni 2021. URL: <https://owasp.org/www-community/Fuzzing> (besucht am 27.03.2022).
- [64] Open Web Application Security Project. *OWASP Top 10:2021*. 30. Sep. 2021. URL: <https://owasp.org/Top10/> (besucht am 15.03.2022).
- [65] Matt Rosenthal. *5 types of cyber security*. 5. Sep. 2018. URL: <https://mind-core.com/blogs/cybersecurity/5-types-of-cyber-security/> (besucht am 15.03.2022).
- [66] Christian Schneider. *Agile Threat Modeling*. 1. Sep. 2021. URL: <https://entwickler.de/security/agile-threat-modeling> (besucht am 04.04.2022).
- [67] Sonarqube. *Code Quality and Code Security*. 1. Jan. 2022. URL: <https://www.sonarqube.org/> (besucht am 10.05.2022).
- [68] Embroker Team. *2021 Must-Know Cyber Attack Statistics and Trends*. 31. Jan. 2021. URL: <https://www.embroker.com/blog/cyber-attack-statistics/> (besucht am 12.12.2021).






Bachelorarbeit_Andreas_Pilgerstorfer

Final Audit Report

2022-06-18

Created:	2022-06-18
By:	Andreas Pilgerstorfer (andreas.pilgi@gmail.com)
Status:	Signed
Transaction ID:	CBJCHBCAABAAjKTJsyUFc2AktqzrO8rICg4GnVEjJ8Rq

"Bachelorarbeit_Andreas_Pilgerstorfer" History

-  Document created by Andreas Pilgerstorfer (andreas.pilgi@gmail.com)
2022-06-18 - 2:01:23 PM GMT- IP address: 93.82.57.145
-  Document emailed to ap.pilgerstorfer@gmail.com for signature
2022-06-18 - 2:02:37 PM GMT
-  Email viewed by ap.pilgerstorfer@gmail.com
2022-06-18 - 2:02:41 PM GMT- IP address: 74.125.216.87
-  Document e-signed by Andreas Pilgerstorfer (ap.pilgerstorfer@gmail.com)
Signature Date: 2022-06-18 - 2:06:16 PM GMT - Time Source: server- IP address: 93.82.57.145
-  Agreement completed.
2022-06-18 - 2:06:16 PM GMT