Sicherheitsaspekte in der (agilen) Softwareentwicklung

FINN HOEDT*, MAURICE RAYMOND PUTZGER*, and BASTIAN WECKE*, Hochschule für Technik, Wirtschaft und Kultur Leipzig (HTWK Leipzig), Deutschland

(Abstract-Länge ist typischerweise 15-25 Zeilen lang, in der PDF-Darstellung)

1 EINLEITUNG UND MOTIVATION

(Beschreibung von Kontext, Problemen, Anforderungen und Zielen)
(kurze Zusammenfassung der Struktur der Belegarbeit)

Diese Arbeit ist folgendermaßen strukturiert. In Kapitel ... ... Abschließend ...

2 GRUNDLAGEN

Im Folgenden werden Begriffe zu den Themen Sicherheit und Shift-Left in der Softwareentwicklung erklärt, um somit eine grundlegende Wissensbasis für die danach folgenden Kapitel zu schaffen.

Als *sichere Software* definieren wir solche, die den grundlegenden Prinzipien der Informationssicherheit [Blakley et al. 2001] entspricht und auch trotz verschiedener Widrigkeiten in ihrer Funktionalität zuverlässig bleibt [Oueslati et al. 2015]. Diese Prinzipien werden in folgende Kategorien unterteilt:

(1) **Vertraulichkeit** beschreibt den Schutz sensibler Daten vor unbefugtem Zugriff.
(2) **Integrität** stellt sicher, dass die Daten konsistent und unverändert bleiben, außer sie werden bewusst durch autorisierte Nutzer verändert.
(3) **Verfügbarkeit** garantiert, dass die Software und ihre genutzten Daten und Ressourcen stets zugänglich sind. Die Verfügbarkeit muss auch unter gezielten Angriffen gewährleistet werden.
(4) **Zuverlässigkeit** leistet ein System dann, wenn es sich zu jedem Zeitpunkt erwartungsgemäß verhält. Dies gilt ebenfalls unter etwaigen Widrigkeiten, wie beispielsweise unbefugten Zugriffen von außen.

Diese vier Prinzipien bilden die Grundlage für sichere Software und müssen somit bei der Entwicklung von sicherheitskritischen Prozessen gezielt beachtet werden.

Der *Shift-Left* Ansatz beschreibt das Verschieben von Prozeduren bzw. Aktivitäten aus späteren Phasen des Software Development Life Cycle in Frühere [Andriadi et al. 2023]. An einer zeitlichen Achse, wie sie in der Abbildung 1 zu sehen ist, lässt sich die Namensherkunft des Shift-Left Ansatzes erklären. Die Abbildung zeigt, wie der Fokus der Softwarequalität von einem späteren Zeitpunkt in einen Früheren verlagert wird. An der Achse entlang gesehen, geschieht diese Verschiebung (engl. Shift) also nach links (engl. left). Diese Idee ist schon länger bekannt, doch der Name Shift-Left für diesen Prozess wurde durch Smith [2001] geprägt. Dieser sprach 2001 vom *Shift-Left Testing* und erklärte, dass durch das frühere Testen in der Softwareentwicklung die Qualität der Software gesteigert und die Kosten von Fehlern gesenkt werden könne [Dawoud et al. 2024]. Dafür müssten Quality Assurance (QA) und Entwickler parallel arbeiten, was zur Folge hätte, dass das QA-Team 1. nicht auf den gesamten Entwicklungsprozess warten muss und 2. entdeckte Fehler bereits während der Entwicklung direkt wieder beseitigt werden können [Andriadi et al. 2023]. Das Ziel dabei ist es Fehler früh zu Erkennen und somit späteren Kosten- oder Zeitaufwand zu sparen [Dawoud et al. 2024]. Auch wenn Shift-Left anfangs für das Testing vorgesehen wurde, lässt das Konzept sich jedoch auf viele erdenkliche Bereiche anwenden.

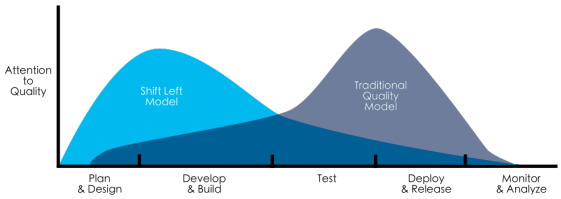

Abb. 1. Shift-Left Verschiebung von Fokus auf Qualität im Software Development Life Cycle

Mit *Shift-Left Security* ist die Anwendung des Shift-Left Gedanken auf den Bereich der Software-Security gemeint [Dawoud et al. 2024]. Viele Sicherheitsprozesse werden in der Regel erst am Ende der Entwicklung durchgeführt, wodurch Fehler oder Sicherheitslücken oft zu spät erkannt werden [Dawoud et al. 2024]. Durch Shift-Left Security werden jene Prozesse in die früheren Phasen des Software Development Cycles verschoben, wodurch Bedrohungen zeitnah erkannt und beseitigt werden können. Jedoch wurde in den eingangs genannten Problemen gezeigt, dass das Einbinden von Sicherheitsprozessen in das agile Arbeiten schwierig umzusetzen ist. Dafür werden im folgenden Konzepte und Methoden definiert, die bei der Umsetzung in agilen Umgebungen helfen. Die Wirksamkeit und genauere Richtlinien dieser Ideen, werden im Hauptteil erläutert.

Development, Security and Operations (DevSecOps) beschreibt die Erweiterung des klassischen *DevOps* durch die Einbindung von Security Aspekten [Rajapakse et al. 2022]. Die Aufgabe des DevSecOps ist es, Sicherheitspraktiken in den gesamten Entwicklungszyklus einzubinden, um so die Sicherheit der Software zu gewährleisten. Dabei werden automatisierte Sicherheitstest kontinuierlich angepasst und ausgewertet, wodurch die Sicherheit gewährleistet wird, sowie auch Zeit und Kosten gespart werden [Lombardi and Fanton 2023]. DevSecOps zielt ebenfalls darauf ab, die Zusammenarbeit und Kommunikation von Entwicklern, Sicherheitsexperten und Betriebsteams zu verbessern, um ein Verständnis von Sicherheit im ganzen Team zu bilden [Rajapakse et al. 2022].

3 (HAUPTTEIL MIT GGF. MEHREREN SECTIONS)

(der Hauptteil umfasst typischerweise ca. 2/3 bis 3/4 des Texts der Arbeit.)

*Alle Studierenden trugen zu gleichen Teilen zu dieser Arbeit bei.

Diese Arbeit wurde im Rahmen des Mastermoduls „Software Engineering“ (Dozent: Prof. Dr. Andreas Both) an der HTWK Leipzig im Wintersemester 2023/2024 erstellt.

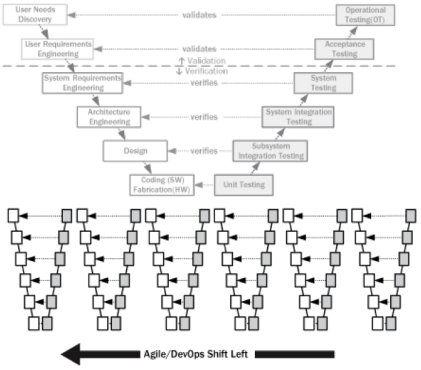

Abb. 2. Agile/DevOps Shift Left (Quelle:)

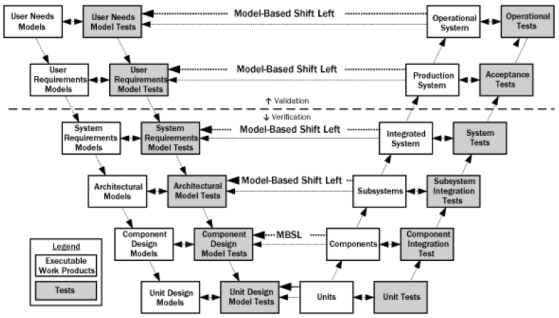

Abb. 3. Model-Based Shift Left (Quelle:)

3.1 Shift-Left Modelle

Der Shift-Left-Gedanke ist eine grundsätzliche Herangehensweise, die sich auf die meisten Konzepte und Modelle anwenden lässt. Da dieser Beitrag sich auf die Verbindung von agilen Methoden und Security durch den Shift-Left-Ansatz konzentriert, werden im Folgenden zwei Shift-Left-Modelle erklärt, die dabei behilflich sind.

Das erste Modell ist das *Agile/DevOps Shift-Left-Testing*. Zur Erklärung des Modells dient die Abbildung 2. Ausgangslage für die Grafik ist das traditionelle V-Modell, welches verblasst in der oberen Hälfte von Abbildung 2 zu erkennen ist. Die mehrfachen Vs darunter visualisieren die Arbeitsweise im agilen Bereich. Jedes V stellt einen Zyklus der Entwicklung dar, der beim agilen Arbeiten meist *Sprint* genannt wird [Rani et al. 2023]. Diese Aufteilung nutzt bereits eine Implementierung des Shift-Left-Ansatzes. Durch das Aufteilen eines gesamten Projekts in mehrere kleine Zyklen werden beispielsweise Tests in jedem Zyklus umgesetzt [Rani et al. 2023]. Betrachtet man die gesamte Laufzeit eines Projekts, werden Tests dadurch automatisch nicht erst am Ende des gesamten Projekts umgesetzt [Bjerke-Gulstuen et al. 2015]. Agile/DevOps Shift-Left-Testing schlägt weiterhin vor, den Fokus auf das Testing in frühere Sprints zu verstärken [Bjerke-Gulstuen et al. 2015]. Dabei ist das Automatisieren dieser Tests ein ausschlaggebendes Prinzip im DevOps-Bereich [Rani et al. 2023]. Shift-Left will diese Automatisierung nutzen, um das kontinuierliche Testen bereits als Bestandteil in die frühen Entwicklungsphasen zu integrieren [Rani et al. 2023]. Die frühzeitige Nutzung von CI/CD und DevOps hilft dabei, Bugs und Sicherheitslücken rechtzeitig zu erkennen und somit langfristige Kosten zu sparen [Rani et al. 2023]. Eine effektive Methode, um Tests für Sicherheitslücken bzw. -probleme zu automatisieren, ist die Nutzung von SAST und DAST, welche später noch genauer analysiert werden. Eine weitere Eigenschaft, die durch die agile Herangehensweise und Shift-Left gestärkt wird, ist die Zusammenarbeit und Kommunikation zwischen Development- und Testing-Teams [Rani et al. 2023]. Durch die Parallelisierung beider Prozesse gibt es einen höheren Austausch von Feedback, wodurch auf Fehler schnell reagiert werden kann. Vor allem bei Projekten mit hohen Sicherheitsanforderungen wird dadurch das Bewusstsein für Sicherheit bei allen Beteiligten gestärkt [Dawoud et al. 2024].

Der zweite Ansatz, der hier vorgestellt werden soll, ist das *Model-Based Shift-Left Testing (MBSLT)*. Wie zuvor veranschaulicht Abbildung 3 die Herangehensweise für das MBSLT basierend auf dem V-Modell. Anders als der Agile/DevOps-Ansatz fokussiert sich MBSLT auf das Testen des gesamten Modells und nicht auf den Quellcode an sich [Rani et al. 2023]. In 3 ist zu erkennen, dass neben jedem standardmäßigen Schritt des V-Modells ein paralleler Schritt erfolgt, der darauf abzielt, das Systemmodell zu testen. Diese Systemmodelle werden beispielsweise durch UML oder Datenflussdiagramme dargestellt [Rani et al. 2023]. Basierend darauf werden Tests erstellt, um Schwachstellen in der Architektur- und Design-Ebene zu erkennen. Auch diese Art von Tests kann durch CI/CD automatisiert werden [Rani et al. 2023]. MBSLT hilft dabei, potenzielle Fehler noch vor der Implementierung zu erkennen, wodurch später Kosten vermieden werden können [Rani et al. 2023]. Des Weiteren wird die Anforderungsanalyse verbessert, da MBSLT sicherstellt, dass Anforderungen vollständig und konsistent sind [Rani et al. 2023]. Dieser Vorteil trägt somit auch zur besseren Kommunikation zwischen dem Development- und dem Testing-Team bei.

3.2 Wirksamkeit von Shift-Left Maßnahmen

In den vorherigen Textabschnitten wurde erklärt, wie Shift-Left in spezifischeren Modellen mit verschiedenen Tools umgesetzt werden kann. Um eine Einschätzung der Wirksamkeit von Shift-Left zu erhalten, wird im Folgenden ein Beitrag von Andriadi et al. [2023] betrachtet.

Andriadi et al. [2023] untersuchten dafür die Menge an Fehlern in einem indonesischen Tech-Unternehmen in den Jahren 2021 und 2022. Das Unternehmen entwickelte zunächst ein Jahr ohne den Shift-Left-Ansatz und im darauffolgenden Jahr mit dem Shift-Left-Ansatz. Die Änderungen, die das Unternehmen 2022 vornahm, beinhalteten hauptsächlich die Arbeitsweise und die Rolle der Entwickler sowie der QA-Teams:

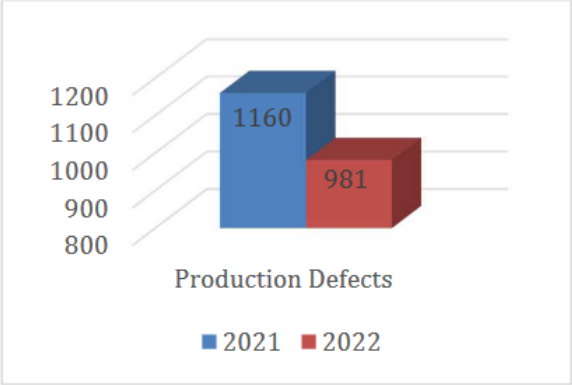

Abb. 4. Diagramm zur Menge an Fehlern im Jahr 2021 und 2022 aus [Andriadi et al. 2023]

(1) Tests werden bereits in der Entwicklungsphase geschrieben und nicht nur in einer gesonderten Testphase.
(2) Entwickler schreiben Basis-Tests für ihre eigenen Features und beheben dadurch entdeckte Fehler eigenständig.
(3) QA erstellt parallel automatisierte Testskripte für den aktuellen Sprint. Die Menge der manuellen Tests wurde reduziert und mehr automatisierte Tests geschrieben.
(4) Entwickler und QA halten gemeinsame tägliche Meetings, um Testanforderungen zu klären. Alle gefundenen Fehler werden dokumentiert und kategorisiert.

In Abbildung 4 ist die Menge der Produktionsfehler in beiden Jahren zu erkennen. Dabei ist festzustellen, dass die Menge an gefundenen Fehlern von 1160 auf 981 gesunken ist. Dies entspricht einer Reduzierung von etwa 15%. Diese Verbesserung wird durch die frühere Erkennung von Fehlern in der Entwicklungsphase begründet, da das Projekt dort weniger Quellcode und Komplexität aufweist [Andriadi et al. 2023]. Ebenfalls konnten die Fehler bei Regressionstests um 24% und bei Integrationstests um 57% reduziert werden. Diese Metriken zeigen, dass der Shift-Left-Ansatz dazu beitragen kann, spätere Kosten zu reduzieren. Der gesamte Prozess brachte jedoch auch Herausforderungen mit sich: Die Entwickler benötigten mehr Zeit, da sie zusätzlich Tests schreiben mussten, und der Entwicklungsprozess war abhängig von einer gut funktionierenden Kommunikation [Andriadi et al. 2023]. Die Ergebnisse zeigen zwar, dass sich dieser Mehraufwand langfristig ausgleicht, dennoch muss der Shift-Left-Ansatz konsequent umgesetzt werden.

Die in der Studie demonstrierte Wirksamkeit von Shift-Left Testing lässt sich analog auf Shift-Left-Security übertragen. Indem Sicherheitsprüfungen bereits in den frühen Entwicklungsphasen integriert werden, lassen sich Risiken minimieren und somit Kosten senken. Die vorgestellte Studie unterstützt dies indirekt, da sie zeigt, dass durch frühzeitige Qualitätssicherung kritische Defekte verhindert werden können – ein Prinzip, das sich auf Sicherheitskontexte übertragen lässt.

3.3 Herausforderungen

In der modernen Softwareentwicklung finden agile Methoden wie Scrum zunehmende Anwendung. Diese Vorgehensweisen versprechen in der Regel eine gesteigerte Produktivität, bessere Produktqualität sowie eine erhöhte Kundenzufriedenheit. Sie zeichnen sich insbesondere durch häufige Änderungen der Anforderungen und kontinuierliche Releases aus. Sicherheitsaspekte sind allerdings in den gängigen agilen Praktiken nicht systematisch verankert. Das erschwert die Entwicklung sicherer Software. [Oueslati et al. 2015]

Im Folgenden werden die wesentlichen Herausforderungen bei der Integration von Sicherheitsanforderungen in agile Entwicklungsprozesse erläutert.

Die Probleme erstrecken sich über verschiedene Phasen des Software Development Life Cycle (SDLC). Ein zentrales Prinzip agiler Methoden ist die kontinuierliche Lieferung von Software. Das bedeutet, dass in jeder Iteration sämtliche SDLC-Aktivitäten durchlaufen werden müssen. Das beinhaltet alle sicherheitsrelevanten Praktiken. Da Iterationen jedoch oft nur wenige Wochen umfassen, ist es schwierig, zeitintensive Sicherheitsmaßnahmen in diesen kurzen Zyklen durchzuführen. [Oueslati et al. 2015]

Darüber hinaus resultieren weitere Herausforderungen aus dem inkrementellen Vorgehen sowie dem Streben nach technischer Exzellenz. Das wird in agilen Methoden unter anderem durch häufiges Refactoring verwirklicht. Die Refactorings können jedoch bestehende Sicherheitsanforderungen versehentlich verletzen. Zusätzlich führt die hohe Priorität, kurzfristige Kundenwünsche umzusetzen, zu Designänderungen. Diese können bestehende Sicherheitsvorgaben brechen und erschweren Sicherheitsmaßnahmen wie Code Reviews. [Oueslati et al. 2015]

Ein weiterer Problembereich betrifft die Qualität der Dokumentation. Nach den agilen Werten hat funktionierende Software Vorrang gegenüber umfassender Dokumentation. Diese Herangehensweise kann zu lückenhafter oder unvollständiger Dokumentation führen. Sicherheitskonzepte erfordern aber in der Regel eine detaillierte und verlässliche Dokumentationsbasis. Tests decken zudem häufig nur die Funktionalität ab und überprüfen somit primär positive Anforderungen. Sicherheit hingegen umfasst zumeist negative Anforderungen, bei denen es darum geht, Fehlverhalten oder Schwachstellen zu erkennen. Da klassische Akzeptanztests hierfür nur bedingt geeignet sind, bleiben potenzielle Sicherheitslücken oft verborgen. [Oueslati et al. 2015]

Des Weiteren entsteht ein Konflikt im Hinblick auf Audits. Agile Methoden sehen einen kontinuierlichen Verbesserungsprozess des Entwicklungsprozesses vor. Audits hingegen verlangen meist einen stabilen, über längere Zeiträume unveränderten Prozess. Dadurch wird Nachvollziehbarkeit und Vergleichbarkeit gewährleistet. Dies erschwert die Einhaltung auditrelevanter Standards in hochflexiblen agilen Umgebungen. [Oueslati et al. 2015]

Das Sicherheitsbewusstsein und die Zusammenarbeit innerhalb des Teams stellen eine weitere Herausforderung dar. Der Projektfortschritt wird in agilen Prozessen vorwiegend an funktionsfähiger Software gemessen. Dadurch rücken Sicherheitsaspekte in den Hintergrund. Darüber hinaus wird empfohlen, sicherheitstechnische Bewertungen unabhängig von den Entwicklerteams durchzuführen.

Somit wird der Einfluss sozialer Beziehungen auf die Ergebnisse vermieden. Dies steht jedoch im Widerspruch zu den agilen Prinzipien, die auf enger Kooperation und fortlaufendem Austausch basieren. [Oueslati et al. 2015]

Schließlich hat auch das Sicherheitsmanagement mit den kurzen Iterationen und dem Fokus auf die Bereitstellung neuer Funktionalitäten zu kämpfen. Umfangreiche Sicherheitsaktivitäten erhöhen die Entwicklungskosten und erfordern zusätzliche Ressourcen. Dazu kommt, dass die Sicherheitsmaßnahmen häufig nicht für die laufende Anwendung erforderlich sind. In vielen Fällen wird daher zugunsten schnellerer Releases auf umfassende Sicherheitsmaßnahmen verzichtet. [Oueslati et al. 2015]

3.4 Static Application Security Testing (SAST)

SAST-Werkzeuge führen eine ”White-Box”-Analyse durch, bei der der Quellcode direkt auf potenzielle Sicherheitslücken untersucht wird. Die Anwendung muss hierfür nicht ausgeführt werden. Durch die frühzeitige Erkennung von Sicherheitslücken in der Entwicklungsphase tragen SAST-Tools maßgeblich dazu bei, potenzielle Schwachstellen proaktiv zu beheben. Typische Beispiele für SAST-Werkzeuge sind Fortify SCA und FindSecurityBugs. [Mateo Tudela et al. 2020]

3.5 Dynamic Application Security Testing (DAST)

DAST-Werkzeuge folgen einem ”Black-Box”-Ansatz, da sie Angriffe von außen auf die laufende Anwendung simulieren. Auf diese Weise lassen sich Sicherheitslücken identifizieren, die erst während der Laufzeit auftreten. DAST-Tools untersuchen primär Schnittstellen einer ausgeführten Anwendung und erfordern keine Kenntnisse über den zugrunde liegenden Quellcode. Daher sind sie in der Regel unabhängig von der verwendeten Programmiersprache. Bekannte Beispiele für DAST-Werkzeuge sind OWASP ZAP und Arachni. [Mateo Tudela et al. 2020]

3.6 Interactive Application Security Testing (IAST

IAST wird, ähnlich wie SAST, dem ”White-Box”-Ansatz zugeordnet, kombiniert jedoch Elemente aus SAST und DAST zu einem hybriden Verfahren. Dabei überwachen und analysieren die IAST-Werkzeuge den Quellcode während der Ausführung, was eine präzise Sicherheitsprüfung anhand realer Laufzeitinformationen ermöglicht. IAST-Werkzeuge sind direkt auf dem Server integriert sind, weshalb sie mit der jeweiligen Programmiersprache kompatibel sein müssen. Da sie ausschließlich serverseitige Analysen durchführen, sind sie nicht in der Lage, clientseitige Sicherheitslücken aufzudecken. Bekannte Beispiele für IAST-Werkzeuge sind Contrast und CxIAST. [Mateo Tudela et al. 2020]

3.7 Effektivität von Application Security Testing

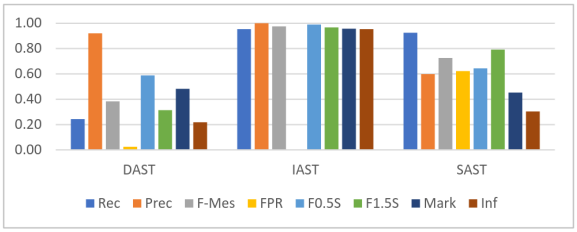

Abb. 5. AST Vergleich ohne Kombination

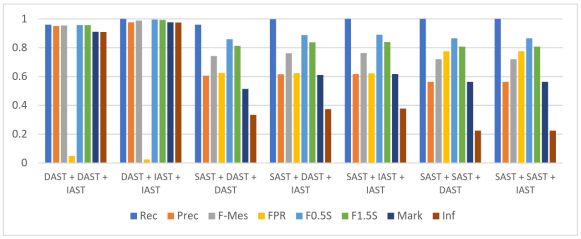

Abb. 6. AST Vergleich mit Kombination

3.8 Frameworks zur Integration von Sicherheit in agile Prozesse

3.8.1 Scrum for Safety. *Scrum for Safety* zielt darauf ab, die Innovationsfähigkeit und Effizienz agiler Methoden mit den strengen Anforderungen an Sicherheit und Konformität zu vereinen. Zu diesem Zweck führt es spezifische Rollen, Prinzipien und Workflows ein, um die Qualität und Sicherheit der zu entwickelnden Software zu gewährleisten. Das Framework erweitert die klassischen Scrum-Rollen um zusätzliche Rollen, die in sicherheitskritischen Projekten erforderlich sind. [Barbareschi et al. 2022] Dazu zählen:

(1) **Verifikator:** Verantwortlich für die Überprüfung der Verifikationstests.
(2) **Validator:** Bestätigt, dass die Anforderungen korrekt und vollständig umgesetzt wurden.
(3) **Assessor:** Externe Person, die sicherstellt, dass der Entwicklungsprozess den geltenden Standards entspricht.

Die zusätzlichen Rollen sind dafür zuständig, die Unabhängigkeit der Prüfprozesse zu gewährleisten und die Erfüllung der Anforderungen an die Softwarequalität sicherzustellen. [Barbareschi et al. 2022]

Neben der Integration zusätzlicher Rollen werden im Rahmen von *Scrum for Safety* auch neue Konzepte eingeführt, die darauf abzielen, die Sicherheit der Softwareentwicklung zu gewährleisten. Hierzu zählt das Sprint-Hardening, dessen Zielsetzung darin besteht, die Bereitstellung einer validierten Softwareversion am Ende jeder Iteration sicherzustellen. In jeder Iteration wird demnach darauf geachtet, dass Benutzerdokumentationen und Konformitätsnachweise erstellt werden, die dann für die externe Softwarebewertung verwendet werden können. [Barbareschi et al. 2022] Ein weiteres Konzept

ist die Continuous Compliance. Das beinhaltet die kontinuierliche Durchführung von Verifizierungs- und Validierungsaktivitäten, sodass die Software jederzeit konform ist. Das Ziel dieses Ansatzes ist die frühzeitige Erkennung und Behebung kritischer Fehler. [Barbareschi et al. 2022] Das dritte Konzept ist die Living Traceability. Die Einführung einer klaren Rückverfolgbarkeit in jedem Prozess bei der Umsetzung von Benutzeranforderungen zielt darauf ab, die Zertifizierung der Software durch externe Prüftstellen zu erleichtern. [Barbareschi et al. 2022]

Der zentrale Prozess in *Scrum for Safety* wird als Safe-Sprint bezeichnet. In Anlehnung an das klassische Scrum-Prinzip stellt der Safe-Sprint eine zeitlich begrenzte Iteration dar, in der ein neues Software-Inkrement entwickelt wird. Im Gegensatz zum klassischen Scrum wird dabei sichergestellt, dass das entwickelte Inkrement durch die eingeführten Konzepte sicherheitstechnisch validiert ist. [Barbareschi et al. 2022] *Scrum4Safety* setzt zu diesem Zweck bereits in der Planungsphase mit dem Safe-Sprint an und integriert Safety-Stories neben den klassischen User-Stories in den Backlog. Das Ziel dieser Vorgehensweise ist die Gewährleistung der Berücksichtigung von Sicherheit bereits in der Anforderungsphase. Eine weitere Maßnahme zur Sicherstellung der Berücksichtigung von Sicherheit in der Anforderungsphase ist die frühzeitige Einbindung von Sicherheitsüberprüfungen und Testphasen im Ablauf des Safe-Sprints. [Barbareschi et al. 2022]

In sicherheitskritischen Projekten nimmt die Dokumentation eine zentrale Rolle ein. Aus diesem Grund wird in den Ablauf eines Safe-Sprints eine Dokumentationsphase integriert (siehe Abbildung 7). Dabei wird Dokumentation nicht als Haupttreiber des Prozesses betrachtet, sondern als Resultat jeder Iteration. [Barbareschi et al. 2022] Das Ziel besteht darin, unnötigen Dokumentationsaufwand zu vermeiden und ausschließlich die für die Zertifizierung erforderlichen Dokumente zu erstellen. *Scrum for Safety* schlägt zur Erstellung von Dokumentationen zusätzlich noch automatisierte Tools vor, wie Doxygen oder IBM DOORS, um Dokumente automatisch aus dem Code zu generieren und den Aufwand zusätzlich zu minimieren. [Barbareschi et al. 2022]

Im Rahmen der Evaluierung von *Scrum for Safety* wurde eine Fallstudie mit Rete Ferroviaria Italiana (RFI) durchgeführt. Gegenstand der Fallstudie war die Entwicklung einer sicheren Middleware für die Kommunikation zwischen Eisenbahnsignalsystemen. Die Herausforderung bestand darin, bestehende Hardware und Protokolle zu unterstützen, um eine nahtlose Integration in vorhandene Systeme zu gewährleisten. Ein weiterer Aspekt, den es zu berücksichtigen galt, war die Sicherstellung der Zuverlässigkeit und Sicherheit der Systeme, um potenzielle Gefahrenquellen zu minimieren und die Einhaltung gesetzlicher Vorgaben sowie branchenspezifischer Richtlinien zu gewährleisten. Die Fallstudie ergab, dass die Integration umfangreicher Sicherheitsmaßnahmen in kurze Iterationen eine Herausforderung darstellte. [Barbareschi et al. 2022] Aus diesem Grund wurde die Länge der Iterationen auf vier Wochen erweitert, wobei etwa 75 % der Sprintzeit für Verifikation und Validierung verwendet wurde.Dies zeigte sich insbesondere bei der Fehlererkennung, da die implementierten Sicherheitsmaßnahmen maßgeblich dazu beitrugen, Fehler frühzeitig zu erkennen und zu beheben. Die Implementierung von Dokumentationsmaßnahmen trug ebenfalls zur Nachvollziehbarkeit der Entwicklungsaktivitäten bei und

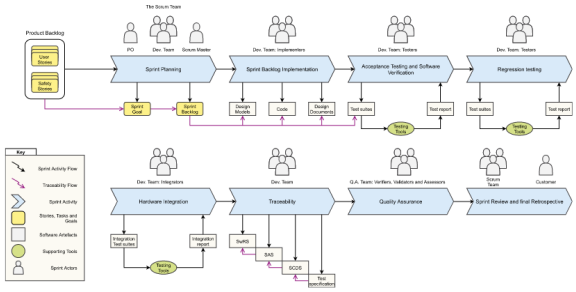

Abb. 7. Scrum for Safety Prozess

erleichterte dadurch die Zertifizierung durch externe Prüfstellen. [Barbareschi et al. 2022]

Insgesamt zeigt die Fallstudie, dass *Scrum for Safety* eine praktikable Lösung bietet, um Sicherheitsanforderungen in agile Entwicklungsprozesse zu integrieren. Die notwendigen Verifikations- und Validierungsmaßnahmen führen zwar zu einer reduzierten Entwicklungsgeschwindigkeit, jedoch bietet das Framework ein solides Grundgerüst, um Sicherheit als zentrales Konzept in der agilen Entwicklung zu verankern. [Barbareschi et al. 2022]

3.8.2 Security Standard Compliant Scaled Agile Framework. S^2C-SAFe ist ein Framework, das auf dem *Scaled Agile Framework (SAFe)* basiert und speziell für sicherheitskritische Projekte in regulatorischen Bereichen entwickelt wurde. Dafür erweitert es das klassische SAFe um Sicherheitsanforderungen aus dem Standard IEC 62443-4-1, der Richtlinien für die sichere Produktentwicklung in industriellen Umgebungen, weshalb es sich für den Gebrauch in großen Entwicklungsteams eignet. [Moyón et al. 2020] Das Ziel ist es, agile Entwicklungsprozesse mit Sicherheitsstandards zu verbinden und dadurch eine kontinuierliche Sicherheitsüberprüfung zu gewährleisten (siehe Abbildung 8), ohne die Flexibilität und Geschwindigkeit der Entwicklung zu beeinträchtigen. [Moyón et al. 2020]

S^2C-SAFe implementiert das Konzept **Continuous Security**, dessen Ziel die Integration von Sicherheit als fundamentalem Aspekt in sämtliche Prozesse ist. Ziel ist es, Sicherheit von einer nichtfunktionalen Anforderung zu einer integralen Komponente der Softwareentwicklung zu machen. Dieses wird bereits in der Planungsphase implementiert, in welcher Sicherheitsanforderungen systematisch in den Backlog integriert werden (siehe Abbildung 8). [Moyón et al. 2020] Hierbei werden Product Owner und Systemarchitekten gezielt in Sicherheitsaspekte eingebunden und entsprechend geschult, um von Anfang an ein hohes Sicherheitsbewusstsein zu gewährleisten. In der Implementierungsphase werden die Sicherheitsanforderungen durch teamübergreifende Coding-Standards umgesetzt und als fester Bestandteil in die Definition of Done integriert. Dies gewährleistet, dass Sicherheitsaspekte bei jeder Entwicklung berücksichtigt werden. [Moyón et al. 2020] Die Qualitätssicherung erfolgt durch dedizierte Verifikations- und Validierungsmaßnahmen, die von unabhängigen Security Experts durchgeführt werden. Ein wesentliches Merkmal des Continuous Security Ansatzes ist dabei die kontinuierliche Durchführung expliziter Sicherheitsanforderungen und Tests, wodurch Sicherheitsmaßnahmen nicht als

einmaliger Check, sondern als fortlaufender Prozess verstanden werden. [Moyón et al. 2020]

Im Rahmen einer Fallstudie wurde *S²C-SAFe* bei Siemens getestet und im Anschluss anhand von Interviews mit 16 Experten aus dem Siemens-Umfeld evaluiert. Die Experten stammen aus verschiedenen Bereichen, darunter Sicherheitsstandards, agile Entwicklung und Compliance. Die Auswertung der Interviews ergab, dass das Framework als praktikabel angesehen wird, um die Diskrepanz zwischen agilen Methoden und Sicherheitsanforderungen zu überbrücken. [Moyón et al. 2020] Darüber hinaus wurde die Visualisierung der Prozesse, die im Rahmen der Entwicklung von *S²C-SAFe* erstellt wurden, als hilfreich erachtet, um eine gemeinsame Sprache zwischen den Teams für Sicherheit und Entwicklung zu etablieren. [Moyón et al. 2020] Trotz der Vorteile, die das Framework für die Integration von Sicherheitsanforderungen in agile Prozesse bietet, sind jedoch auch Herausforderungen zu verzeichnen. Insbesondere die mangelnde Sicherheitsexpertise innerhalb der agilen Teams führte zu Schwierigkeiten bei der Implementierung der Sicherheitsanforderungen. Dies zeigte sich insbesondere bei der Priorisierung von Sicherheitsanforderungen, wobei häufig unklar war, welche Sicherheitsmaßnahmen am dringendsten umgesetzt werden sollten. [Moyón et al. 2020] Darüber hinaus erschwerte die unterschiedliche Interpretation von Sicherheitsanforderungen durch Entwickler und Sicherheitsexperten das Prozessmanagement und könnte zu Missverständnissen und Fehlern führen. Die Autoren schlagen daher die Einführung spezifischer Rollen wie eines Security Product Owner oder eines Secure System Architect vor, um die Kommunikation zwischen den Teams zu optimieren und die Umsetzung von Sicherheitsanforderungen zu erleichtern. [Moyón et al. 2020] Eine weitere Herausforderung stellt die Komplexität der Sicherheitsanforderungen dar, da die Integration der Maßnahmen in die kurzen Entwicklungszyklen als herausforderd zu bewerten ist. [Moyón et al. 2020]

Zusammenfassend lässt sich feststellen, dass *S²C-SAFe* eine praktikable Lösung bietet, um Sicherheitsanforderungen in großen agilen Entwicklungsprojekten zu integrieren. Es hilft, die Kluft zwischen agilen Methoden und regulatorischen Anforderungen zu überbrücken und fördert die Zusammenarbeit zwischen Sicherheits- und Entwicklungsteams. [Moyón et al. 2020] Die implementierung von kontinuierlichen Sicherheitsmaßnahmen in die Prozesse der Softwareentwicklung wird durch die Anwendung der Konzepte aus dem *Shift-Left*-Ansatzes im Framework unterstützt, wodurch Sicherheitsaspekte von Beginn an in den Entwicklungszyklus integriert werden.

4 DISKUSSION

(Einordnung, Interpretation und Bewertung der Erkenntnisse – (nachvollziehbare, begründbare) Meinungen sind erlaubt)

5 ZUSAMMENFASSUNG UND AUSBLICK

(Überblick über die gesamte Arbeit, Rückführung auf Aussagen aus Kapitel 1 durchführen, offene Punkte als neue Forschungsfragen definieren)

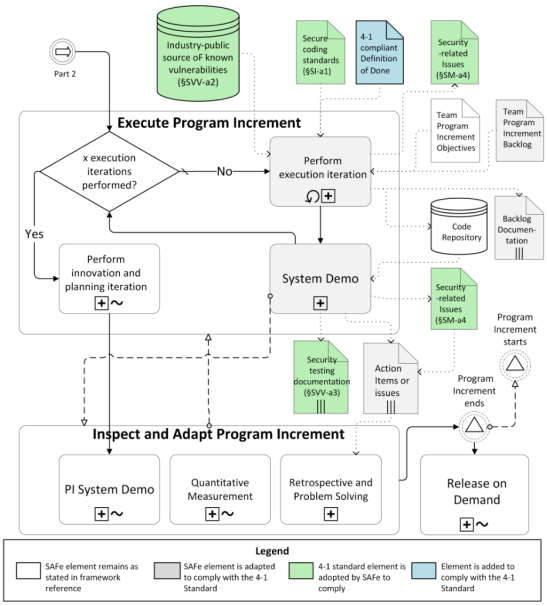

Abb. 8. Security Standard Compliant Scaled Agile Framework

LITERATUR

Kus Andriadi, Haryono Soeparno, Ford Lumban Gaol, and Yulyani Arifin. 2023. The Impact of Shift-Left Testing to Software Quality in Agile Methodology: A Case Study. In *2023 International Conference on Information Management and Technology (ICIMTech)*. 259–264. https://doi.org/10.1109/ICIMTech59029.2023.10277919 ISSN: 2837-2778.

Mario Barbareschi, Salvatore Barone, Riccardo Carbone, and Valentina Casola. 2022. Scrum for safety: an agile methodology for safety-critical software systems. *Software Quality Journal* 30, 4 (Dec. 2022), 1067–1088. https://doi.org/10.1007/s11219-022-09593-2

Kristian Bjerke-Gulstuen, Emil Wiik Larsen, Tor Stålhane, and Torgeir Dingsøyr. 2015. High Level Test Driven Development – Shift Left. In *Agile Processes in Software Engineering and Extreme Programming*, Casper Lassenius, Torgeir Dingsøyr, and Maria Paasivaara (Eds.). Springer International Publishing, Cham, 239–247. https://doi.org/10.1007/978-3-319-18612-2_23

Bob Blakley, Ellen McDermott, and Dan Geer. 2001. Information security is information risk management. In *Proceedings of the 2001 workshop on New security paradigms*. ACM, Cloudcroft New Mexico, 97–104. https://doi.org/10.1145/508171.508187

Abdallah Dawoud, Soeren Finster, Nicolas Coppik, and Virendra Ashiwal. 2024. Better Left Shift Security! Framework for Secure Software Development. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 642–649. https://doi.org/10.1109/EuroSPW61312.2024.00078 ISSN: 2768-0657.

Federico Lombardi and Alberto Fanton. 2023. From DevOps to DevSecOps is not enough. CyberDevOps: an extreme shifting-left architecture to bring cybersecurity within software security lifecycle pipeline. *Software Quality Journal* 31, 2 (June 2023), 619–654. https://doi.org/10.1007/s11219-023-09619-3

Francesc Mateo Tudela, Juan-Ramón Bermejo Higuera, Javier Bermejo Higuera, Juan-Antonio Sicilia Montalvo, and Michael I. Argyros. 2020. On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications. *Applied Sciences* (Jan. 2020). https://doi.org/10.3390/app10249119 Number: 24 Publisher: Multidisciplinary Digital Publishing Institute.

Fabiola Moyón, Daniel Méndez Fernández, Kristian Beckers, and Sebastian Klepper. 2020. How to Integrate Security Compliance Requirements with Agile Software Engineering at Scale? 69–87. https://doi.org/10.1007/978-3-030-64148-1_5

Hela Oueslati, Mohammad Masudur Rahman, and Lotfi ben Othmane. 2015. Literature Review of the Challenges of Developing Secure Software Using the Agile Approach. In *2015 10th International Conference on Availability, Reliability and Security*. 540–547. https://doi.org/10.1109/ARES.2015.69

Roshan N. Rajapakse, Mansooreh Zahedi, M. Ali Babar, and Haifeng Shen. 2022. Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology* 141 (Jan. 2022), 106700. https://doi.org/10.1016/j.infsof.

2021.106700

V Shobha Rani, Dr A Ramesh Babu, K. Deepthi, and Vallem Ranadheer Reddy. 2023. Shift-Left Testing in DevOps: A Study of Benefits, Challenges, and Best Practices. In *2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS)*. 1675–1680. https://doi.org/10.1109/ICACRS58579.2023.10404436

Larry Smith. 2001. Shift-Left Testing. http://www.drdobbs.com/shift-left-testing/184404768

A ANHANG 1