Sicherheitsaspekte in der (agilen) Softwareentwicklung

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

Die Anwendung agiler Entwicklungsmethoden hat in den letzten Jahren stark zugenommen. Es hat sich gezeigt, dass der Einsatz agiler Methoden zu einer gesteigerten Produktivität, einer erhöhten Produktqualität und einer verbesserten Kundenzufriedenheit führt. Allerdings zeigen sich in der Praxis häufig Schwierigkeiten, da traditionelle Sicherheitsprozesse nicht mit der Geschwindigkeit und Flexibilität agiler Methoden vereinbar sind. Dies hat zur Folge, dass Sicherheitsaspekte im Bereich der Softwareentwicklung häufig vernachlässigt werden. Gegenstand der vorliegenden Arbeit ist die Untersuchung eines konkreten Anwendungsfalls des Shift-Left-Ansatzes, dessen Ziel darin besteht, Sicherheitsmaßnahmen frühzeitig in den Entwicklungsprozess zu integrieren. Ziel ist es, sicherheitskritische Fehler frühzeitig zu erkennen und zu beheben. Im Rahmen der Untersuchung werden verschiedene Shift-Left-Strategien analysiert und deren Wirksamkeit sowie Herausforderungen diskutiert. Darüber hinaus werden konkrete Tools und Frameworks zur Sicherheitsintegration in agile Prozesse präsentiert und deren Ergebnisse in der praktischen Anwendung analysiert. Die Untersuchung ergibt, dass der Einsatz von Shift-Left-Praktiken zu einer verbesserten Integration von Sicherheitsaspekten in agile Prozesse führen kann. Dennoch bestehen weiterhin Herausforderungen, insbesondere bei der Integration zeitintensiver Sicherheitsmaßnahmen.

1 EINLEITUNG UND MOTIVATION

In der modernen Softwareentwicklung finden agile Methoden wie Scrum zunehmend Anwendung. Häufig wird dabei ein Zusammenhang zwischen gesteigerter Produktivität, verbesserter Produktqualität und erhöhter Kundenzufriedenheit hergestellt. Ein wesentliches Merkmal dieser Methoden ist die hohe Flexibilität, die durch häufige Änderungen der Anforderungen und kontinuierliche Releases charakterisiert ist. Jedoch ist zu kritisieren, dass Sicherheitsaspekte in den gängigen agilen Praktiken nicht systematisch verankert sind. [Nägele et al. 2023] Dies erschwert die Entwicklung sicherer Software. [Oueslati et al. 2015] Die in der Regel nur wenige Wochen umfassenden Iterationen in agilen Prozessen lassen die Durchführung zeitintensiver Sicherheitsmaßnahmen in diesen kurzen Zyklen nicht zu. Dies führt zu erhöhten Entwicklungskosten und dem Bedarf zusätzlicher Ressourcen. In vielen Fällen wird daher zugunsten schnellerer Releases auf umfassende Sicherheitsmaßnahmen verzichtet. [Oueslati et al. 2015] Gegenstand der vorliegenden Arbeit ist demnach die Untersuchung des Einsatzes des Shift-Left-Ansatzes in Bezug auf Sicherheit in agilen Prozessen.

Zu diesem Zweck werden zunächst die Grundlagen des Shift-Left-Ansatzes erläutert und die verschiedenen Arten von Shift-Left-Praktiken vorgestellt und deren Wirksamkeit dargelegt. Im Anschluss werden konkrete Tools und Frameworks zur Integration von Sicherheit in agile Prozesse präsentiert und deren Ergebnisse in der praktischen Anwendung analysiert. Ziel ist es, Handlungsmöglichkeiten für die Integration von Shift-Left-Praktiken aufzuzeigen und die Wirksamkeit dieser Ansätze zu bewerten. Ein besonderes Augenmerk gilt dabei den Herausforderungen bei der Integration von Sicherheitsmaßnahmen in agile Prozesse und den diskutierten Lösungsansätzen.

2 GRUNDLAGEN

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

*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.

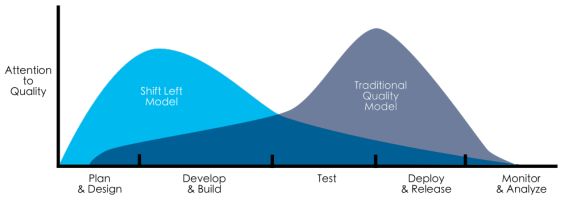

Abb. 1. Shift-Left-Verschiebung von Fokus auf Qualität im Software Development Life Cycle [Andriadi et al. 2023]

vorgesehen wurde, lässt sich das Konzept jedoch auf viele denkbare Bereiche anwenden.

Mit *Shift-Left Security* ist die Anwendung des Shift-Left-Gedankens 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 Sicherheitstests 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 verankern [Rajapakse et al. 2022].

3 HAUPTTEIL

In den kommenden Kapiteln werden konkrete Ideen, Modelle und Tools zum Shift-Left-Ansatz vorgestellt, mit einem Fokus auf Sicherheit in agilen Prozessen. Ziel ist es die theoretischen Konzepte mit praktischer Anwendung zu verbinden. Der Hauptteil baut somit auf den vorangegangenen Definitionen und Grundlagen auf und zielt darauf ab, Handlungsmöglichkeiten für die Integration von Shift-Left-Praktiken aufzuzeigen.

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.

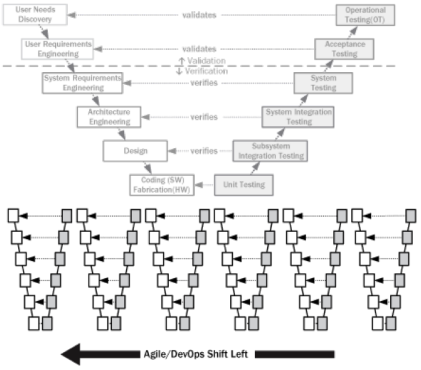

Abb. 2. Agile/DevOps Shift Left [Firesmith 2015]

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 implementiert [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, Fehler 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 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 an 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

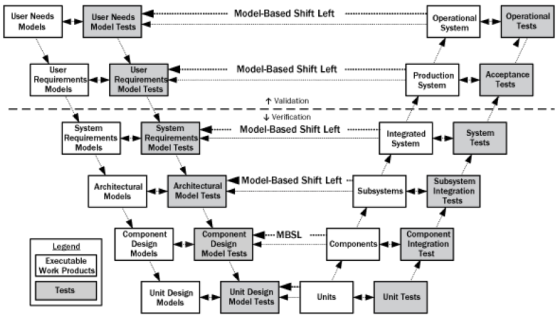

Abb. 3. Model-Based Shift Left [Firesmith 2015]

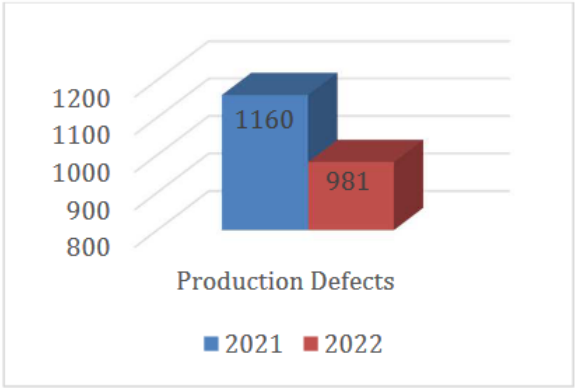

Abb. 4. Diagramm zur Menge an Fehlern im Jahr 2021 und 2022 [Andriadi et al. 2023]

des Projektes [Rani et al. 2023]. In Abbildung 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 spätere 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-, Testing- und Security-Team bei.

3.2 Wirksamkeit von Shift-Left Maßnahmen

In den vorherigen Textabschnitten wurde erklärt, wie Shift-Left in unterschiedlichen Ebenen eines Projekts mit verschiedenen Tools genutzt 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 an einem Projekt 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:

(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 zu schreiben.
(4) Entwickler und QA halten gemeinsam tägliche Meetings, um Testanforderungen zu klären. Alle gefundenen Fehler werden dokumentiert und kategorisiert.

Nach den zwei Jahren Entwicklung wurde die Menge an gefundenen Fehlern in beiden Jahren verglichen und überprüft ob eine Verringerung stattgefunden hat. 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 den zeitigen Entwicklungsphase begründet, da das Projekt dort weniger Quellcode und eine geringere Komplexität aufweist [Andriadi et al. 2023]. Ebenfalls konnten die Fehler bei Regressionstests um 24% und bei Integrationstests um 57% reduziert werden. Des Weiteren wurde die Anzahl an Hotfixes von 43 auf 25 verringert, was einer Reduzierung von 42% entspricht. Alle diese Metriken zeigen, dass der genutzte Shift-Left-Ansatz dazu beigetragen hat, die Softwarequalität zu steigern und spätere Kosten zu reduzieren. Andriadi et al. [2023] stellten in diesen Zuge fest, das der Prozess jedoch auch Herausforderungen mit sich bringt: Die Entwickler benötigten mehr Zeit, da sie zusätzlich Tests schreiben mussten, und der Entwicklungsprozess war abhängig von einer gut funktionierenden Kommunikation aller Beteiligten [Andriadi et al. 2023]. Diese Feststellung zeigt, dass der Shift-Left-Ansatz effektiv ist, gleichzeitig aber gut geplant und richtig umgesetzt werden muss.

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 Application Security Testing (AST)

Im Rahmen von Shift-Left-Ansätzen wird angestrebt, Test- und Qualitätssicherungsmaßnahmen möglichst früh in den Softwareentwicklungsprozess einzubinden. Besonders im Bereich sicherheitsrelevanter Tests erweist sich diese Strategie als äußerst effektiv, da Schwachstellen auf diese Weise bereits in frühen Phasen entdeckt und behoben werden können. Automatisierte Verfahren spielen dabei eine große Rolle, da sie sich nahtlos in bestehende CI/CD-Pipelines integrieren lassen. [Rani et al. 2023] Eine wesentliche Komponente in diesem Kontext stellt das Application Security Testing (AST) dar. Unter diesem Begriff werden verschiedene Methoden zusammengefasst, die auf die Identifizierung von Sicherheitslücken in Softwareanwendungen abzielen. [Mateo Tudela et al. 2020] In den folgenden Abschnitten werden drei wesentliche AST-Methoden näher betrachtet.

3.3.1 Static Application Security Testing (SAST). Static Application Security Testing (SAST) Werkzeuge führen eine White-Box-Analyse durch, bei der der Quellcode direkt auf potenzielle Sicherheitslücken untersucht wird. Die Ausführung der Anwendung ist dafür nicht erforderlich. Diese Analyse erlaubt frühzeitige Erkennung von Sicherheitslücken in der Entwicklungsphase. Daher tragen SAST-Tools dazu bei, potenzielle Schwachstellen proaktiv zu beheben. [Mateo Tudela et al. 2020] Im Gegensatz zur dynamischen Analyse kann SAST auch hard-to-reach States erfassen, also Codepfade, die während der Laufzeit nur selten oder unter speziellen Bedingungen auftreten. SAST-Werkzeuge arbeiten mit vordefinierten Regeln und Mustern, um typische Schwachstellen wie Pufferüberläufe, unsichere API-Nutzung oder unzureichende Eingabevalidierung zu erkennen. Dabei kommen unterschiedliche methodische Ansätze zum Einsatz, die von einfachen Pattern-Matches bis hin zu komplexen, semantischen Codeanalysen mithilfe von Abstract Syntax Trees (ASTs) reichen. Ein wesentlicher Nachteil von SAST-Tools besteht darin, dass sie ausschließlich Schwachstellen aufdecken können, für die entsprechende Erkennungsmuster existieren. Zudem neigen sie vergleichsweise häufig zu falsche Positiv- oder Negativmeldungen. Daher sollten sie als ergänzende Sicherheitsmaßnahme in Kombination mit anderen Tests eingesetzt werden. [Chess and McGraw 2004]

3.3.2 Dynamic Application Security Testing (DAST). Dynamic Application Security Testing (DAST) basiert auf einer Black-Box-Analyse, bei der Angriffe auf eine laufende Anwendung simuliert werden. Dadurch lassen sich insbesondere Sicherheitslücken identifizieren, die erst während der Ausführung auftreten. Durch das gezielte Senden von Payloads wird überprüft, inwieweit eine Anwendung Angriffen wie Cross-Site Scripting (XSS) oder SQL Injection (SQLi) standhält. Im Gegensatz zu SAST werden somit reale Angriffszenarien erfasst. [Singh et al. 2024]

Ein wesentlicher Vorteil von DAST besteht darin, dass die eingesetzten Werkzeuge keine Kenntnisse über den Quellcode benötigen und damit unabhängig von der verwendeten Programmiersprache sind. Allerdings lassen sich DAST-Verfahren erst in späteren Phasen des SDLC einsetzen, da eine lauffähige Anwendung vorausgesetzt wird. Nachdem die Tools konfiguriert und das Zielsystem festgelegt wurde, umfassen die typischen Schritte einer DAST-Analyse: [Singh et al. 2024]

(1) **Crawling**: Die DAST-Tools simulieren Nutzerinteraktionen mit der Anwendung und crawlen durch deren Seiten und Funktionen. Dabei identifizieren sie sämtliche Ein- und Ausgabepunkte, um die Struktur der Anwendung zu erfassen.
(2) **Simulation von Angriffen**: Die Tools generieren schadhafte Payloads (Command Injection oder Reflected XSS) und senden diese an die zuvor entdeckten Eingabepunkte.
(3) **Identifizierung von Schwachstellen**: Die Responses der Anwendung werden analysiert, um Schwachstellen wie unsichere Sicherheitskonfigurationen oder mangelhafte Eingabevalidierungen zu entdecken.
(4) **Berichterstattung**: Abschließend wird ein detaillierter Report der erkannten Sicherheitslücken erstellt, der häufig auch Informationen zu potenziellen Gegenmaßnahmen sowie den Schweregrad der jeweiligen Schwachstelle enthält.

3.3.3 Interactive Application Security Testing (IAST). Interactive Application Security Testing (IAST) wird ähnlich wie SAST, dem White-Box-Ansatz zugeordnet. Es kombiniert jedoch die Vorteile aus SAST und DAST zu einem hybriden Verfahren. Dabei überwachen und analysieren IAST-Werkzeuge den Quellcode während der Laufzeit, wodurch eine präzisere Sicherheitsbewertung auf Basis realer Laufzeitinformationen ermöglicht wird. [Mateo Tudela et al. 2020]

Im Gegensatz zu rein statischen oder dynamischen Testmethoden bietet IAST eine kontextbezogene Analyse, da es sowohl die Quellcode-Struktur als auch das tatsächliche Laufzeitverhalten der Anwendung berücksichtigt. Dies führt zu einer höheren Genauigkeit bei der Erkennung von Schwachstellen. IAST-Werkzeuge sind direkt auf dem Server integriert und müssen daher mit der jeweiligen Programmiersprache kompatibel sein. Allerdings ist IAST auf serverseitige Analysen beschränkt und kann daher keine clientseitigen Schwachstellen, aufdecken. [Pan 2019]

3.4 Effektivität von Application Security Testing

Es stellt sich die Frage, in welchem Umfang die Methoden SAST, DAST und IAST tatsächlich bei der Identifizierung von Sicherheitslücken unterstützen. Der Artikel Mateo Tudela et al. [2020] befasste sich mit der Effizienz dieser Verfahren und untersuchte, welche Kombination die besten Ergebnisse liefert.

Als Testgrundlage diente das OWASP Benchmark Project, eine in Java geschriebene Open-Source-Webanwendung, die auf Apache Tomcat läuft. Dieses Projekt enthält schwerwiegende Sicherheitslücken entsprechend den OWASP Top 10 von 2013 und 2017. Insgesamt umfasst die Anwendung 320 Testfälle, von denen die Hälfte keine echten Schwachstellen sind. [Mateo Tudela et al. 2020]

In der Studie kamen drei Open-Source- und drei kommerzielle AST-Tools zum Einsatz: [Mateo Tudela et al. 2020]

- SAST: Fortify SCA und FindSecurityBugs
- DAST: OWASP ZAP und Arachni
- Contrast Community Edition und CxIAST

Zur Bewertung der Verfahren wurden etablierte Metriken herangezogen. True Positives (TP) bezeichnen tatsächliche Schwachstellen, die auch gefunden wurden, False Positives (FP) sind fälschlicherweise erkannte Sicherheitslücken, und False Negatives (FN) reale Schwachstellen, die übersehen wurden. Auf Basis dieser Werte lassen sich verschiedene Kennzahlen berechnen: [Mateo Tudela et al. 2020]

- **Recall**: Anteil der tatsächlich vorhandenen Schwachstellen, die korrekt erkannt werden.
- **Precision**: Verhältnis der korrekt erkannten Schwachstellen (TP) zu allen gemeldeten Schwachstellen (TP + FP).
- **True Positive Rate (TPR)**: Verhältnis der gefundenen zu allen existierenden Schwachstellen.
- **False Positive Rate (FPR)**: Verhältnis der falsch erkannten Schwachstellen (FP) zu allen nicht vorhandenen Schwachstellen (TN).
- **F-Measure**: Harmonischer Mittelwert von Precision und Recall.
- **$F_ß$-Score**: Spezifische Variante der F-Measure, bei der das Verhältnis zwischen Recall und Precision über ß unterschiedlich gewichtet wird. Dabei legt $ß = 0,5$ mehr Gewicht auf Precision, während $ß = 1,5$ den Recall stärker berücksichtigt.
- **Markedness**: Verhältnis der positiven Vorhersagen unter Berücksichtigung von FP und FN.
- **Informedness**: Betrachtung aller positiven und negativen Fälle. Jeder erkannte Schwachstellen (TP) erhöht diesen Wert um $1/P$ (wobei P die Anzahl positiver Fälle ist), während jeder nicht entdeckte Fall (FN) den Wert um $1/N$ eduziert (wobei N die Anzahl negativer Fälle ist).

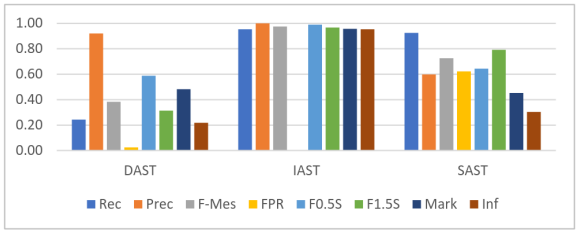

Abb. 5. AST Vergleich ohne Kombination [Mateo Tudela et al. 2020]

In Abbildung 5 wird die Effektivität bei der Erkennung der OWASP-Top-10-Schwachstellen für jeden AST-Typ ohne Kombination dargestellt. IAST-Tools erzielen hier in allen Metriken die besten Werte. Sie nähern sich dem Idealwert (1) an und weisen gleichzeitig die niedrigste FPR (0) auf. SAST-Tools liefern ein insgesamt gutes Ergebnis, verzeichnen jedoch einen relativ hohen FPR-Wert von etwa 0,60. Im Vergleich dazu schneiden DAST-Tools bei der FPR deutlich besser ab, weisen jedoch nur einen geringen Recall von circa 0,20 auf. Die Kombination aus niedrigem Recall und vergleichsweise hohem TPR resultiert in einer hohen Precision (ca. 0,90). [Mateo Tudela et al. 2020]

Daraus lässt sich ableiten, dass eine Kombination aus DAST und SAST potenziell bessere Resultate liefern könnte. Eine Verbindung aus SAST und IAST würde zwar die TPR weiter erhöhen, jedoch auch mehr FP erzeugen. Eine Kombination aus IAST und DAST würde von der niedrigen FPR seitens DAST profitieren. [Mateo Tudela et al. 2020]

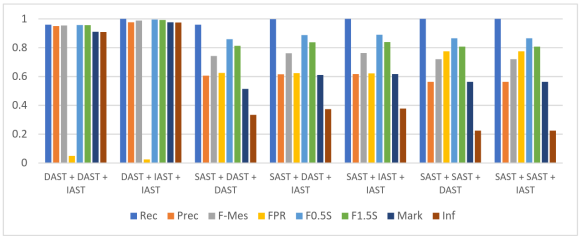

Abb. 6. AST Vergleich mit Kombination [Mateo Tudela et al. 2020]

Im nächsten Schritt wurden verschiedene Zweier- und anschließend Dreierkombinationen untersucht, wie in Abbildung 6 ersichtlich. Die Dreierkombinationen liefern ähnliche Ergebnisse wie die Zweierkombinationen. Am besten schnitt eine Kombination aus zwei IAST-Tools und einem DAST-Tool ab, gefolgt von einer Kombination aus einem IAST-Tool und zwei DAST-Tools. Beide Konfigurationen erreichen sehr gute Werte für Recall und TPR (zwischen 0,95 und 1). Die übrigen Kombinationen verzeichnen eine höhere FPR und somit in den anderen Metriken niedrigere Werte. [Mateo Tudela et al. 2020]

Welche Kombination für ein Unternehmen die beste Wahl darstellt, hängt stark von dem Sicherheitskritischem Level der jeweiligen Anwendung ab. Der Artikel differenziert hier zwischen hoher, mittlerer und niedriger Sicherheitskritikalität der Anwendungen. [Mateo Tudela et al. 2020]

Für Webanwendungen, die als hochkritisch eingestuft werden und hochsensible Daten verarbeiten, empfiehlt es sich, jeden Alarm im Detail zu untersuchen. Bei der Wahl der AST-Tools sollte daher der Fokus auf einem hohen TPR liegen. False Positives müssen dabei in Kauf genommen werden, um möglichst alle Schwachstellen zu identifizieren. In mittelkritischen Umgebungen sollte neben einer hohen TPR vor allem auf eine hohe Precision geachtet werden. Es soll ein Übermaß an FP vermieden werden, um den Aufwand für nachgelagerte Analysen zu reduzieren. Bei niedrigkritischen Webanwendungen, in denen keine oder nur gering empfindliche Daten verarbeitet werden, ist eine hohe TPR bei gleichzeitig geringer FPR anzustreben. Es ist akzeptabel, dass nicht jede potenzielle Schwachstelle entdeckt wird, solange der Zeit- und Ressourcenaufwand für die Prüfung von Fehlalarmen möglichst gering bleibt. Somit ist ein hoher Markedness-Score eine hilfreiche Orientierungsgröße. [Mateo Tudela et al. 2020]

Insgesamt wird durch die Ergebnisse des Artikels verdeutlicht, dass der Einsatz und die gezielte Kombination von AST-Tools eine äußerst effektive Methode zur Identifizierung von Sicherheitslücken ist. Insbesondere IAST-Tools liefern hervorragende Resultate und nehmen damit eine große Rolle bei der automatisierten Schwachstellenerkennung ein. Sie unterstützen sowohl Entwickler als auch Unternehmen maßgeblich dabei, sichere Software zu erstellen. [Mateo Tudela et al. 2020]

3.5 Frameworks zur Integration von Sicherheit in agile Prozesse

Die Zunahme von Sicherheitsrisiken stellt eine der signifikantesten Herausforderungen für Organisationen dar, wobei die Hauptquelle für Sicherheitsschwachstellen im Bereich der Softwareentwicklung liegt. Die Diskrepanz zwischen den Anforderungen an Sicherheit und den agilen Methoden ist evident, da die traditionellen Sicherheitsprozesse häufig nicht mit der agilen Entwicklungsgeschwindigkeit Schritt halten können. [Nägele et al. 2023] Um dieser Herausforderung zu begegnen, wurden verschiedene Frameworks entwickelt, die darauf abzielen, Sicherheitsaspekte in agile Prozesse zu integrieren. Dabei finden sich Konzepte des Shift-Left-Ansatzes wieder, die eine frühzeitige Einbindung von Sicherheitsmaßnahmen in den Entwicklungsprozess ermöglichen. Im Folgenden werden zwei dieser Frameworks vorgestellt und es erfolgt eine Diskussion ihrer jeweiligen Vor- und Nachteile.

3.5.1 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 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]

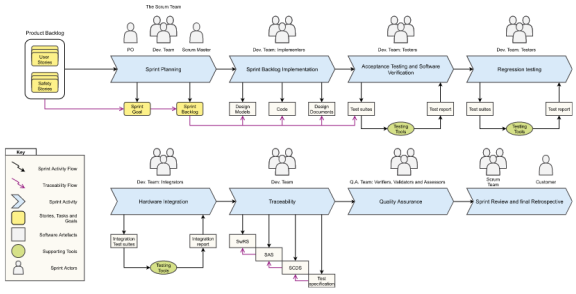

Abb. 7. Scrum for Safety Prozess [Barbareschi et al. 2022]

3.5.2 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 S^2C-*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 S^2C-*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 S^2C-*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

Die vorgestellten Modelle, Tools und Studien zeigten verschiedene Herangehensweisen und Umsetzungsmöglichkeiten des Shift-Left-Ansatzes. Jedes Konzept nutzte Shift-Left, um die Qualität und Sicherheit in Softwareprojekten zu stärken. Die daraus gewonnenen Erkenntnisse gilt es nun kritisch zu betrachten, um anschließend das Potenzial von Shift-Left realistisch einzuschätzen.

Es ist festzustellen, dass Shift-Left ein übergreifendes Konzept ist, das sich auf viele verschiedene Aspekte der Softwareentwicklung anwenden lässt [Andriadi et al. 2023]. Die Motivation für Shift-Left ergibt sich aus der Erkenntnis, dass späte Änderungen am Code teuer sind und ein frühes Beheben von Fehlern Kosten sowie Zeit spart. Diese Einsicht ist nicht neu und wurde bereits in den 1980er-Jahren von Boehm [1984] ausführlich diskutiert. Die Idee, früh zu testen, ist damit älter als der Begriff Shift-Left selbst, der erst durch [Smith 2001] geprägt wurde [Dawoud et al. 2024]. Dennoch schmälert das nicht die Wirksamkeit des Konzepts.

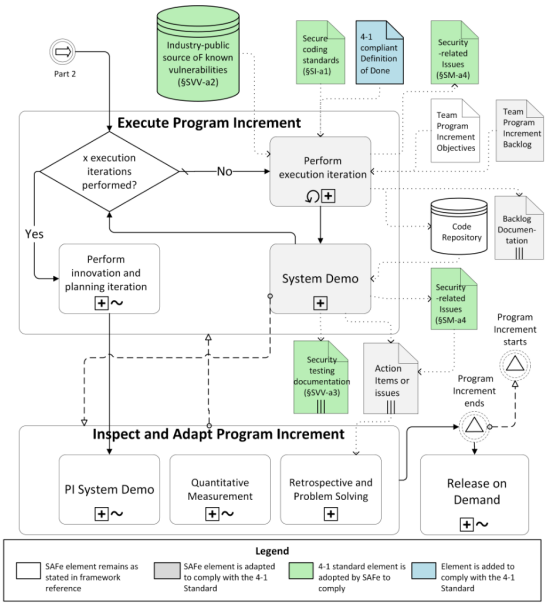

Abb. 8. Security Standard Compliant Scaled Agile Framework [Moyón et al. 2020]

Die vielfältige Anwendbarkeit von Shift-Left zeigt sich in den verschiedenen Formen dieses Ansatzes (3.1). So kann Shift-Left sowohl auf Quellcode-Ebene genutzt werden, um den Entwicklungsprozess zu beeinflussen, als auch auf der Planungsebene, beispielsweise durch MBSLT. Eine detaillierte Anwendung dieser Konzepte liefert die Studie von [Andriadi et al. 2023], die die Effektivität des Shift-Left-Ansatzes belegt. In dem untersuchten Projekt wurden Testing-Aktivitäten und Abläufe nach links verschoben sowie die Kommunikation zwischen Entwickler- und QA-Teams verbessert. Dies führte zu einer gesteigerten Softwarequalität und einer Reduzierung von Fehlern in der Anwendung. Durch Shift-Left konnte in diesem Fall die Qualität des Projekts gesichert werden.

Weiterhin zeigen Konzepte wie SAST, DAST und IAST, dass sich Sicherheitsüberprüfungen automatisieren und somit ebenfalls nach links verschieben lassen. Die Effektivität dieser Tools wurde anhand der Untersuchung von Mateo Tudela et al. [2020] dargelegt. Durch die Nutzung von SAST, DAST und IAST können Sicherheitsprüfungen besser in agile Entwicklungsprozesse integriert werden, sodass Sicherheitslücken frühzeitig erkannt und behoben werden können.

Beispiele für eine strukturierte Anwendung von Shift-Left liefern die Frameworks Scrum4Safety und S2C-SAFe [Barbareschi et al. 2022] [Moyón et al. 2020]. Diese bieten einen strukturierten Ansatz für die agile Entwicklung sicherheitskritischer Software und kombinieren somit Agilität und Sicherheitsanforderungen effektiv. In beiden Modellen ist Shift-Left in verschiedenen Bereichen erkennbar: Neben dem frühzeitigen Testing auf Code-Ebene werden auch abstrakte Elemente wie Safety Stories eingeführt, um Sicherheitsanforderungen bereits vor der Entwicklung zu festigen.

Die Modelle und Tools zeigen, dass Shift-Left ein sinnvoller und effektiver Ansatz zur Verbesserung der Softwarequalität und -sicherheit ist. Allerdings kommt Shift-Left-Security nicht ohne Herausforderungen. Beispielsweise stellten Andriadi et al. [2023] fest, dass durch die frühzeitige Integration von Sicherheitstests die Entwickler mehr Zeit benötigten und eine intensivere Kommunikation zwischen Entwickler- und QA-Teams erforderlich war. Dies erhöht sowohl die Kosten als auch die Komplexität der Umsetzung.

Ähnliches lässt sich in den Frameworks Scrum4Safety und S2C-SAFe beobachten. Im Vergleich zu rein agilen Modellen erfordern sie stärker strukturierte Prozesse und eine klarere Rollenverteilung. Für Teams, die vollständig agil arbeiten, kann dies einschränkend wirken. Zwar ermöglichen die Frameworks eine Kombination von Agilität und Sicherheit, jedoch nicht nahtlos und ohne zusätzliche Aufwände. Der verstärkte Fokus auf Sicherheit führt zu einer gewissen Einschränkung des Arbeitsprozesses und erfordert geschulte Entwickler.

Das zeigt, dass Shift-Left-Security zwar ein sinnvoller und effektiver Ansatz ist, aber keine Universallösung darstellt. Der Schlüssel liegt in einer sorgfältigen Implementierung und einer ausführlichen Analyse im Vorfeld. Jedes sicherheitskritische Projekt sollte gezielt geplant werden, um zu bestimmen, wie Shift-Left optimal integriert werden kann. Dafür können etablierte Frameworks wie Scrum4Safety oder S2C-SAFe genutzt werden – vorausgesetzt, es ist bekannt, dass sie für den jeweiligen Kontext eine sinnvolle Lösung darstellen.

5 ZUSAMMENFASSUNG UND AUSBLICK

Die agile Softwareentwicklung zeichnet sich durch kurze Entwicklungszyklen, häufige Releases und sich schnell ändernde Anforderungen aus. Diese Flexibilität beschleunigt die Entwicklung, erschwert jedoch die Implementierung umfassender Sicherheitsmaßnahmen. Insbesondere traditionelle Sicherheitsansätze sind schwer mit agilen Methoden vereinbar, da sie oft auf langfristige Planungen und umfangreiche Prüfprozesse setzen. [Oueslati et al. 2015]

Um dieses Problem zu adressieren, wurde der Shift-Left-Ansatz vorgestellt, um sicherheitsrelevante Aufgaben frühzeitig in den Entwicklungsprozess integriert. [Dawoud et al. 2024] In diesem Zusammenhang wurden die zwei Methoden Agile/DevOps Shift-Left-Testing und Based Shift-Left-Testing vorgestellt. Die beiden Methoden unterscheiden sich dabei in ihrem Ansatz. Agile/DevOps Shift-Left-Testing zielt darauf ab, Tests frühzeitig in den Entwicklungsprozess zu integrieren und den Quellcode durch automatisierte Tests in jedem Sprint zu überprüfen. [Rani et al. 2023] Model-Based Shift-Left-Testing (MBSLT) hingegen fokussiert sich auf das Testen des Systemmodells, um Architektur- und Designfehler bereits in der frühen Planungsphase zu identifizieren, beispielsweise mithilfe von UML- oder Datenflussdiagrammen. [Rani et al. 2023]

Ergänzend dazu wurde das Application Security Testing (AST) als gezielte Strategie zur Vorverlagerung sicherheitsrelevanter Tests betrachtet. Besonders effektiv erwies sich dabei die Kombination verschiedener SAST-, DAST- und IAST-Tools, die unterschiedliche Schwachstellen frühzeitig identifizieren. [Mateo Tudela et al. 2020]

Darüber hinaus wurden zwei erweiterte Frameworks untersucht, die den Shift-Left-Ansatz übernehmen und um branchenspezifische Anforderungen ergänzen. Scrum for Safety erweitert das klassische

Scrum-Framework durch zusätzliche Sicherheitsrollen und Safe-Sprints, wodurch Sicherheitsaspekte kontinuierlich geprüft werden. [Barbareschi et al. 2022]

S^2C-SAFe überträgt diese Prinzipien auf groß angelegte Projekte und ermöglicht so eine konsistente Sicherheitsstrategie auch in komplexen Entwicklungsumgebungen. Beide Ansätze verdeutlichen, dass eine klare Rollenverteilung, durchgängige Compliance-Prüfungen und automatisierte Dokumentation essenziell für eine erfolgreiche Sicherheitsintegration sind. Gleichzeitig bleibt die Verbindung von Agilität und Sicherheit eine anspruchsvolle Aufgabe, die sowohl tiefgehendes technisches Wissen als auch eine geeignete Team- und Prozesskultur erfordert. [Moyón et al. 2020]

Eine universelle Lösung zur nahtlosen Integration von Sicherheitsmaßnahmen in agile Entwicklungsprozesse existiert bislang nicht. Jedes der vorgestellten Modelle besitzt spezifische Vor- und Nachteile, die je nach Anwendungskontext unterschiedlich wirken. Zukünftig ist zu erwarten, dass diese Konzepte weiter optimiert werden, um ihre Anwendung zu erleichtern und flexibler an verschiedene Einsatzszenarien anzupassen.

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

Barry W. Boehm. 1984. Software Engineering Economics. *IEEE Transactions on Software Engineering* SE-10, 1 (Jan. 1984), 4–21. https://doi.org/10.1109/TSE.1984.5010193 Conference Name: IEEE Transactions on Software Engineering.

B. Chess and G. McGraw. 2004. Static analysis for security. *IEEE Security & Privacy* 2, 6 (Nov. 2004), 76–79. https://doi.org/10.1109/MSP.2004.111 Conference Name: IEEE Security & Privacy.

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.

Donald Firesmith. 2015. Four Types of Shift Left Testing. https://web.archive.org/web/20150905082941/https://insights.sei.cmu.edu/sei_blog/2015/03/four-types-of-shift-left-testing.html

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

Sascha Nägele, Nathalie Schenk, and Florian Matthes. 2023. The Current State of Security Governance and Compliance in Large-Scale Agile Development: A Systematic Literature Review and Interview Study. In *2023 IEEE 25th Conference on Business Informatics (CBI)*. 1–10. https://doi.org/10.1109/CBI58679.2023.10187439 ISSN: 2378-1971.

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

Yuanyuan Pan. 2019. Interactive Application Security Testing. In *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*. 558–561. https://doi.org/10.1109/ICSGEA.2019.00131

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

Ravinder Singh, Mukesh Kumar Gupta, Dipak Raghunath Patil, and Sarang Maruti Patil. 2024. Analysis of Web Application Vulnerabilities using Dynamic Application Security Testing. In *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)*. 1–6. https://doi.org/10.1109/I2CT61223.2024.10543484

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