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** ist 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, kann man die Verschiebung nach Links erkennen. Die Abbildung zeigt, wie der Fokus von der Softwarequalität von einem späteren Zeitpunkt in einen Früheren verlagert wird. Dieses Vorgehen ist schon länger bekannt, doch der Name Shift-Left 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 die Entwicklung warten muss und 2. entdeckte Fehler bereits während der Entwicklung direkt wieder beseitigt werden können [Andriadi et al. 2023].

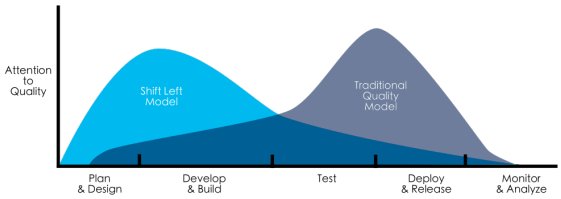

Abb. 1. Shift-Left Verschiebung von Fokus auf Qualität im Software Development Life Cycle

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.

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. 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.)

3.1 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]

*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.

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]

3.2 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]

3.3 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]

3.4 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]

3.5 Effektivität von Application Security Testing

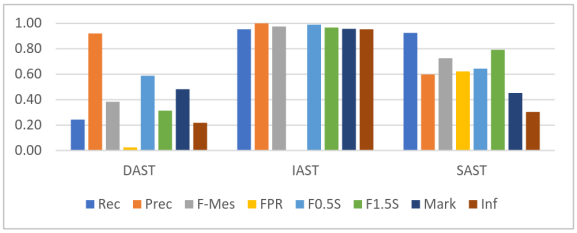

Abb. 2. Scrum4Safety Prozess

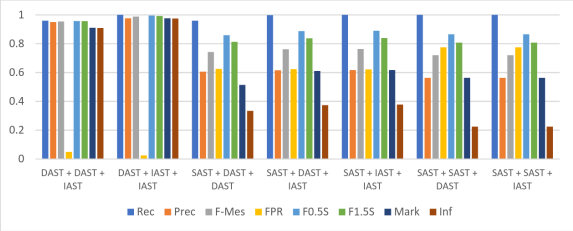

Abb. 3. Scrum4Safety Prozess

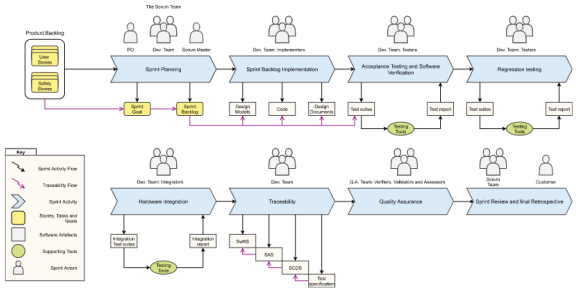

Abb. 4. Scrum4Safety Prozess

3.6 Scrum4Safety

Scrum4Safety verfolgt das Ziel, die Innovationsfähigkeit und Effizenz von agilen Methoden mit den strengen Anforderungen an Sicherheit und Konformität zu kombinieren. Dafür führt es spezifische Rollen, Prinzipien und Workflows ein, um die Qualität und Sicherheit der zu entwickelnden Software sicherzustellen. Das Framework erweitert die klassischen Scrum-Rollen um zusätzliche Rollen, die in sicherheitskritischen Projekten notwendig sind. Dabei handelt es sich um Verifier, Validator und Assessor. Diese Rollen sind für den Quality-Assurance-Prozess verantwortlich und überprüfen die Einhaltung der Sicherheitsanforderungen.

Neben den zusätzlichen Rollen führt Scrum4Safety auch neue Konzepte ein, die die Sicherheit der Softwareentwicklung gewährleisten sollen. Dazu gehört das Sprint-Hardening, welches sicherstellen soll, dass am Ende jeder Iteration eine validierte Softwareversion bereitsgestellt werden kann. Dafür wird in jeder Iteration darauf geachtet, dass Benutzerdokumentationen und Konformitätsnachweise erstellt werden, die dann für die externe Softwarebewertung verwendet werden können. Ein weiteres Konzept ist die Continuous Compliance. Das beinhaltet, dass kontinuierlich Verfifizierungs- und Validierungsaktivitäten durchgeführt werden, sodass die Software jederzeit konform ist. Dadurch sollen kritische Fehler frühzeitig erkannt werden, um sie schnellstmöglich beheben zu können. Das dritte Konzept ist die Living Traceability. Durch die Einführung einer klaren Rückverfolgbarkeit in jedem Prozess bei der Umsetzung von Benutzeranforderungen soll die Zertifizierung der Software durch externe Prüftstellen erleichert werden. [Andriadi et al. 2023]

Der zentrale Prozess in Scrum4Safety ist der sogennante Safe-Sprint. Dabei handelt es sich wie beim klassischen Scrum-Prinzip um eine zeitlich begrenzte Iteration, in der ein neues Software-Inkrement entwickelt wird. Im Gegensatz zum klassischen Scrum soll das entwickelte Inkrement aber durch die eingeführten Konzepte sicherheitstechnisch validiert sein. Dafür setzt Scrum4Safety mit dem Safe-Sprint schon in der Planning-Phase an und integriert Safety-Stories neben den klassischen User-Stories in den Backlog. Dadurch soll gewährleistet sein, dass Sicherheit bereits in der Anforderungsphase berücksichtigt wird. Auch eine frühe Einbindung von Sicherheitsüberprüfungen und Testphasen im Ablauf des Safe-Sprints soll sicherstellen, dass Sicherheitsmaßnahmen frühzeitig in den Entwicklungsprozess 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)

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.

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.

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.

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

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

A ANHANG 1