Ausarbeitung Softwarequalität

Software-Qualitätsbewertung mittels Software Metriken

Softwarequalität
Duale Hochschule Baden-Württemberg
Stuttgart

von
Paul Walker und Leon Kampwerth

Matrikelnummer: 3610783, 5722356
Abgabedatum: 22.03.2023

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abstract

Softwarequalität zu messen kann sich mitunter als schwierig herausstellen. Diese Arbeit hat das Ziel einfache Metriken für Qualitätscharakteristika zu erarbeiten. Diese Metriken sollen einfach automatisiert messbar sein und so einen kontinuierlichen Indikator für die Softwarequalität geben, der auch in Continuous Integration / Continuous Delivery (CI/CD) Prozesse genutzt werden kann. Der Fokus wird auf die Qualitätscharakteristika Wartbarkeit und Sicherheit gesetzt.

1 Einleitung

Die meisten Menschen arbeiten jeden Tag mit verschiedener Software. In der Arbeit mit Office-Software oder je nach Beruf spezialisierter Software für bestimmte Aufgaben und/oder Maschinen. Zu Hause nutzen viele Menschen Unterhaltungssoftware wie Spiele, oder Videoplattformen. Software ist heutzutage allgegenwärtig und die meisten Menschen, werden auch schon einmal Software schlechter Qualität genutzt haben. Das kann durch abruptes Schließen einer Applikation, unintuitives Design oder langer Ladezeiten für den Nutzer erkennbar sein. Es kann recht einfach sein, schlechte Qualitäten in Software zu erkennen, allerdings lässt sich Softwarequalität so nicht Objektiv vergleichen. Diese Arbeit beschäftigt sich damit Metriken zu finden, mit denen Software Qualität Objektiv gemessen werden kann.

2 Softwarequalität

Um Softwarequalität messen zu können muss zuerst festgelegt werden, was Softwarequalität überhaupt ist. Viele Aspekte von Softwarequalität sind für den Endnutzer gar nicht erkennbar oder sind nur dann erkennbar, wenn dieser Aspekt eine schlechte Qualität hat.

In dieser Arbeit wird die Definition von Softwarequalität nach International Organization for Standardization (ISO) 25010 [7] verwendet. In dem ISO 25010 Standard wird ein Produktqualitätsmodell für System und Softwarequalität. Nach diesem Modell wird die Qualität eines Softwaresystems oder Produkts durch acht Charakteristika definiert. Diese Charakteristika werden in Abbildung 1 dargestellt.

Jedes dieser Charakteristika wird weiter in Subcharakteristika aufgeteilt. In dieser Arbeit wird der Fokus auf zwei bestimmte Charakteristika gesetzt und für jedes eine Metrik vorgestellt, mit der die Qualität der Charakteristik gemessen werden kann.

Eine einzelne Metrik für Softwarequalität zu erstellen ist schwer, da die Softwarequalität durch so viele Charakteristika beeinflusst wird. Außerdem würde eine einzelne Metrik für die Softwarequalität nur bedingt nützlich sein. Es wäre zwar möglich die generelle Qualität einer Software zu quantifizieren, allerdings lässt sich dann nicht herausfinden, in welchen Bereichen Mängel an der Qualität der Software vorliegen. Es ist sinnvoller einzelne Metriken für die Qualitätscharakteristika zu erstellen, wie es auch in dieser

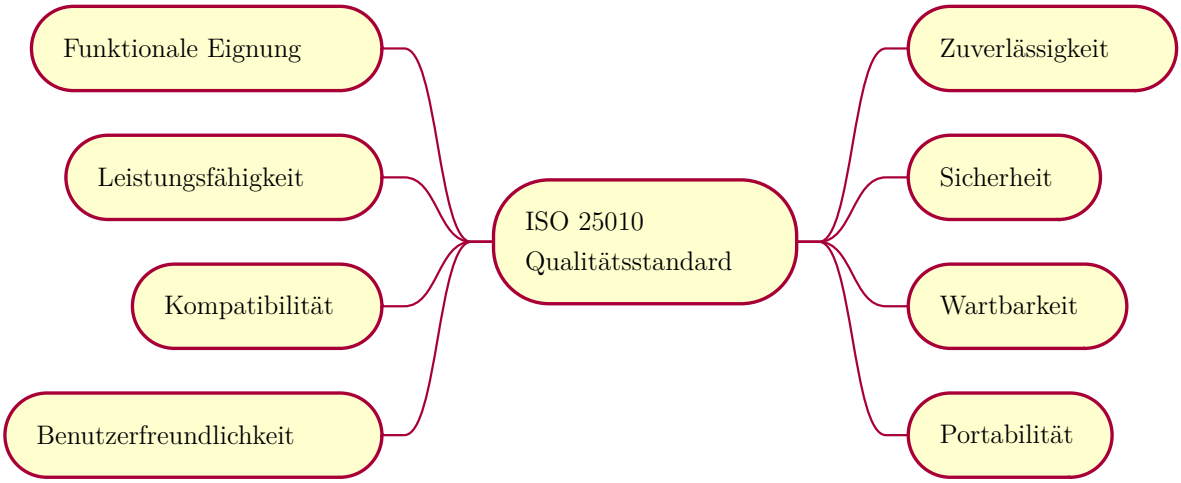

Abbildung 1: Softwarequalität nach ISO 25010 [7]

Arbeit gemacht wird, da so die Ursache für potenzielle Qualitätsmängel einfacher gefunden werden kann.

2.1 Wartbarkeit

Der Grad der Wartbarkeit ist nach ISO, wie effektiv und effizient das Produkt oder System verändert werden kann [7]. Diese Veränderungen können Korrekturen, Verbesserungen oder Anpassungen der Software auf Änderungen des Einsatzgebiets oder auf Änderungen der Anforderungen sein [7]. Die Wartbarkeit bezieht sich auch auf die Installation von Updates [7]. Die Wartbarkeit ist nach ISO in weitere Subcharakteristika aufgeteilt. Diese Subcharakteristika sind in Abbildung 2 dargestellt.

Die Wartbarkeit ist eine der wichtigsten Qualitätscharakteristika für Entwickler. Viele andere Qualitätscharakteristika wie die Benutzerfreundlichkeit oder die Leistungsfähigkeit bewerten äußere Eigenschaften, die für den Entwicklungsprozess nicht generell eine Rolle spielen. So ist die Leistungsfähigkeit von Software meist nur in kritischen Bereichen der Software relevant. Die Benutzerfreundlichkeit von Software spielt in der Regel nur in Bereichen, mit denen ein Benutzer interagiert eine Rolle. Dagegen ist die Wartbarkeit von Software immer dann relevant, wenn an und mit bestehendem Quellcode gearbeitet werden muss. Softwareprodukte oder Systeme, die eine hohe Wartbarkeit aufweisen, werden in der Entwicklung und der Wartung einfacher sein als Softwareprodukte oder Systeme, die eine geringe Wartbarkeit aufweisen. Außerdem muss während des ganzen Entwicklungsprozess darauf geachtet werden die potenziell geltenden Regeln und Vorschriften zur Wartbarkeit von Software einzuhalten.

2.1.1 Modularität

Die Modularität wird daran gemessen, inwieweit das System oder Programm aus einzelnen Teilen besteht und Änderungen an einem Teil der Software andere Teile nur

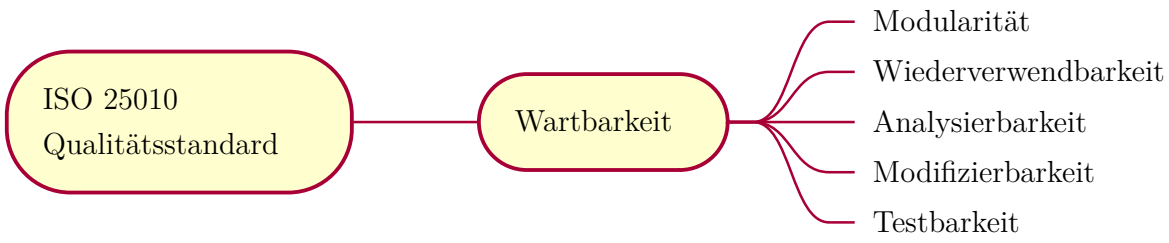

Abbildung 2: Wartbarkeit nach ISO 25010 [7]

minimal oder gar nicht beeinflussen [7]. Module sind Softwareattribute, die Software in unabhängige Einzelteile trennen [11].

2.1.2 Wiederverwendbarkeit

Die Wiederverwendbarkeit ist der Grad, zu dem ein Teil des Systems in mehr als einem System oder beim Erstellen von weiteren Teilen des Systems verwendet werden kann [7]. Die Wiederverwendung bezieht sich dabei nicht nur auf Software, sondern auch auf Dokumentation, Software-Design, Applikationstests und weitere Teile des Systems [6].

2.1.3 Analysierbarkeit

Wie effektiv und effizient es möglich ist, den Einfluss auf ein Produkt oder System abzuschätzen, wenn ein oder mehrere Teile verändert werden, ist durch die Analysierbarkeit definiert [7]. Außerdem wie gut der Ursprung von Fehlern im Produkt oder System diagnostiziert werden kann [7].

2.1.4 Modifizierbarkeit

Die Modifizierbarkeit ist ein Maßstab dafür wie effektiv und effizient sich ein Produkt oder System ändern lässt, ohne dabei Defekte einzuführen oder die Produktqualität zu verschlechtern [7].

Die Modifizierbarkeit kann durch die *Modularität* und die *Analysierbarkeit* beeinflusst werden [7].

2.1.5 Testbarkeit

Der Grad der Testbarkeit ist, wie effektiv und effizient Testkriterien für ein Produkt oder System definiert werden können und wie effektiv und effizient Tests für das Produkt oder System durchgeführt werden können [7]. Zusätzlich zur Erstellung der Testkriterien und der Durchführung der Tests, spielt es auch eine Rolle wie gut objektive und messbare Tests erstellt werden können, die feststellen, ob ein bestimmtes Kriterium erfüllt wurde [11].

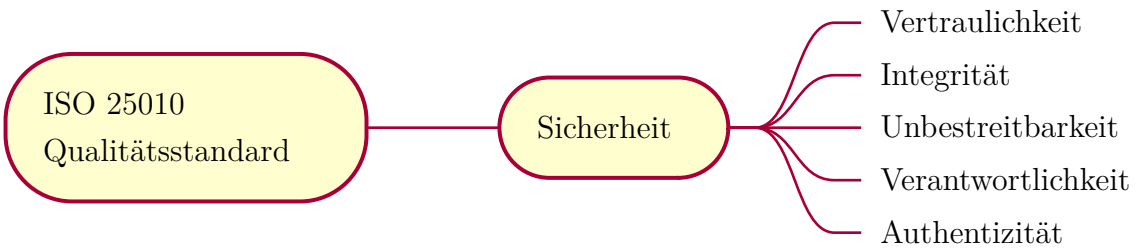

Abbildung 3: Sicherheit nach ISO 25010 [7]

2.2 Sicherheit

Sicherheit beschreibt nach ISO, wie gut ein Produkt oder ein System die Informationen und Daten schützt. Ziel ist es sicherzustellen, dass Personen und andere Systeme nur den Grad an Zugriff auf die Daten haben, für den sie berechtigt sind [7]. Dies bezieht sich sowohl auf die Daten, die vom System gespeichert werden, als auch auf die Daten, die vom System übertragen werden [7]. Sicherheit ist ein wichtiges Qualitätsmerkmal für Software und verbessert das Vertrauen der Benutzer und der Stakeholder in die Software [7]. Die Sicherheit ist nach ISO in weitere Subcharakteristika aufgeteilt. Diese Subcharakteristika sind in Abbildung 3 dargestellt.

Ähnlich zur Wartbarkeit ist auch Sicherheit ein sehr wichtiges Qualitätscharakteristika. Sicherheit wird jedoch nur dann bemerkt wird, wenn sie abwesend ist und die Folgen fehlender Sicherheit zeigen sich meist erst einige Zeit nach der Entwicklungsphase. Daher hat die Sicherheit der Software für viele Entwickler eher eine geringere Priorität und die Erfahrung mit dem Themenbereich ist eher gering [15] [16]. Trotz einem starken Anstieg an Hackerattacken (insbesondre durch den Einsatz von Ransomware) in den letzten Jahren [2] [4] ist Sicherheit weiterhin kein festes Kriterium in der Design- und Entwicklungsphase vieler Teams. Stattdessen wird die Sicherheit meist kurz vor dem Ende der Entwicklung bedacht [15], was die Möglichkeit zum Beheben von architekturellen oder groben Entwicklungsfehlern sehr schwer macht. Um die Qualität der Sicherheit einer Software zu erhöhen, muss sie schon vor der eigentlichen Entwicklung, in der Designphase, mit bedacht werden. Ähnlich zur Wartbarkeit gilt es Regeln und Vorschriften zu definieren, die über die Entwicklung hinweg eingehalten werden.

2.2.1 Vertraulichkeit

Vertraulichkeit misst, wie gut das System sicherstellen kann, dass ein Zugriff auf Daten nur von berechtigten Benutzern durchgeführt werden kann [7].

2.2.2 Integrität

Die Integrität eines Systems beschreibt wie gut es den unauthorisierten Zugriff oder die Modifikation auf seine Programme oder Daten verhindern kann [7].

2.2.3 Unbestreitbarkeit

Der Grad der Unbestreitbarkeit beschreibt, wie gut ein System beweisen kann, dass eine Aktion oder ein Ereignis stattgefunden hat. So soll es einer Entität nicht möglich sein, dies zu bestreiten [7].

Nach der ISO gibt es zwei Möglichkeit dies zu erreichen. Der Ursprungsnachweis ist die erste Möglichkeit. Hier erhällt das System (der Empfänger) einen Nachweis, wer die Aktion oder das Ereignis ausgelöst hat. Dies verhindert, dass eine Entität bestreiten kann, dass sie die Aktion oder das Ereignis ausgelöst hat [9]. Die zweite Möglichkeit ist der Zustellnachweis. Hier erhält die Entität, die die Aktion oder das Ereignis ausgelöst hat, einen Nachweis, dass das System (der Empfänger) die Aktion oder das Ereignis durchgeführt hat. Dies verhindert, dass das System bestreiten kann, dass es die Anweisungen erhalten hat [9]. Beide Möglchkeiten können einzeln eingesetzt werden oder auch kombiniert werden.

2.2.4 Verantwortlichkeit

Verantwortlichkeit gibt an wie gut die Aktionen einer Entität (z.B. Person, System) auf dieses zurückgeführt werden können [7]. Die Verantwortlichkeit kann durch die *Unnachgibigkeit* und die *Integrität* des Systems beeinflusst werden.

2.2.5 Authentizität

Der Grad der Gewissheit, mit dem die Identität einer Entität (z.B. Person, System) bewiesen werden kann wird durch die Authentizität definiert [7]. Das Ziel ist es mit Sicherheit nachweisen zu können, das eine Entität auch die ist, die sie angibt zu sein [8].

3 Software Metriken

Software Metriken sind quantitative Maße, die zur Bewertung von Softwareprodukten und Softwaresystemen genutzt werden können. Diese Metriken können verschiedene Aspekte der Software Bewerten. Hier werden Metriken vorgestellt, mit denen die Wartbarkeit und die Sicherheit analysiert werden könne.

Software Metriken geben sowohl den Entwicklern als auch den Stakeholdern der Software Informationen über Eigenschaften der Software. Diese Informationen können nützlich sein, um Risiken frühzeitig zu erkennen, Verbesserungen und Verschlechterungen zu quantifizieren und verschiedene Software vergleichen zu können. Software Metriken können in der Kommunikation zwischen Entwicklern und Stakeholdern helfen, da sie helfen Eigenschaften der Software zu objektifizierten und sich so beispielsweise bestimmte Entwicklungstätigkeiten, wie das Refactoring von Software besser gegenüber den Stakeholdern rechtfertigen lassen.

Mit Softwaremetriken die, die Softwarequalität quantifizieren können Regeln eingeführt werden, die bestimmte Mindestanforderungen an die Softwarequalität stellen. So kann in der Theorie eine konstante objektiv gute Qualität der Software gewährleistet werden.

Um diese Regeln umzusetzen ist es hilfreich die nötigen Metriken automatisch zu generieren und Änderungen an der Software nur dann zu erlauben, wenn die Regeln eingehalten werden.

3.1 Wartbarkeit

Viele Charakteristika der Wartbarkeit lassen sich über die Dauer oder Effizienz von Wartungsaufgaben messen. Es kann beispielsweise die Zeit gemessen werden, die benötigt wird, um eine Änderung an der Software durchzuführen, um damit die Modifizierbarkeit zu charakterisieren [5].

Diese Metriken lassen sich allerdings nur bei konkreten Wartungsaufgaben messen. Das ist problematisch, da so die Qualität der Software nur durch großen Aufwand bestimmt werden kann. Ziel hier ist es daher eine Metrik zu finden, mit der sich die Wartbarkeit ohne großen Aufwand bestimmen lässt. Das Paper *A Practical Model for Measuring Maintainability* von Ilja Heitlager, Tobias Kuipers und Joost Visser [5] stellt eine Möglichkeit dar, die Wartbarkeit ohne großen Aufwand zu messen. Allerdings basiert diese Paper auf dem älteren ISO 9126 Qualitätsmodell [5, 10]. In dieser Arbeit wird die Metrik, die in [5] daher erweitert und auf das neuere ISO 25010 Qualitätsmodell angepasst.

Für diese Metrik werden einige einfache Kennzahlen der Software kombiniert, um Kennwerte für die Jeweiligen Subcharakteristika der Wartbarkeit zu berechnen. Diese Kennwerte können dann verwendet werden, um einen Gesamtwert für die Wartbarkeit der Software zu bekommen. Die Kennzahlen die Genutzt werden sind.

Quellcodevolumen

Die Zahl der effektiven Quellcodezeilen im gesamten Bereich der analysiert werden soll. Je geringer das Quellcodevolumen ist, desto einfacher ist es eine Übersicht über den Bereich zu bekommen, der analysiert werden soll. Also ist niedriger besser.

Komplexität der Einheiten

Die zyklomatische Komplexität einer Einheit sagt aus wie Komplex eine Einheit ist, und damit auch wie komplex Wartungsaufgaben sein können. Eine Einheit ist dabei das kleinste ausführbare Stück Code [5]. Je nach Programmiersprache kann eine Einheit eine Funktion, eine Klasse oder aber das gesamte Programm sein [5]. Eine geringere Komplexität spricht für besser wartbare Software. Die zyklomatische Komplexität von Software kann mit Tools wie *scc* [1] automatisiert gemessen werden. Hier ist also niedriger besser.

Quellcode Duplikation

Quellcode Duplikation ist definiert durch die Anzahl Quellcode Zeilen, die eins zu eins an verschiedenen Stellen zu finden sind. Duplikation spricht für Probleme in der Software. Also ist für eine höhere Qualität der Software eine niedrige Duplikation besser.

Einheitengröße

Eine geringe Einheitengröße sorgt für eine höhere Softwarequalität, da kleinere Einheiten einfacher zu warten sind.

Unit-Testabdeckung

Die Testabdeckung durch Unit-Tests wird häufig verwendet, um Aussagen über die Softwarequalität zu machen. Hier wird diese Metrik auch verwendet. Allgemein gilt, je mehr Unit-Tests vorhanden sind, desto besser.

Modulanzahl

Die Modulanzahl gibt Aussage über die Modularisierung von Software. Je höher die Modulanzahl, desto modularisierter ist die Software. Hier ist also höher besser.

Wiederverwendung von Einheiten

Die Wiederverwendung von Einheiten sagt aus, wie oft eine Einheit an anderer Stelle im Quellcode wiederverwendet wird. Das kann gemessen werde, indem gezählt wird wie oft Klassen genutzt werden oder wie oft Funktionen aufgerufen werden. Hier gilt, höher ist besser.

Diese Kennzahlen werden wie folgt kombiniert, um Metriken für die Subcharakteristika zu erhalten.

Modularität

Eine einfache Metrik für die Modularität ist die Modulanzahl in dem betrachteten Teil der Software. Nachteil dieser Metrik ist, dass sie nicht aussagt, inwieweit Änderungen an einem Modul die anderen Module beeinflussen. Nach ISO 25010 sollen Änderungen an einem Modul, die anderen Module möglichst wenig beeinflussen [7].

Eine andere Metrik ist die Einheitengröße. Diese gibt auch eine gewisse objektive Aussage über die Modularität. Für eine hohe Modularität ist eine geringe Kopplung und hohe Kohäsion, also hohe Zusammengehörigkeit, innerhalb der Einheiten nötig um zu gewährleisten, dass Änderungen an einem Modul, die anderen Module minimal oder gar nicht beeinflussen. Viele kleine Einheiten sprechen für eine hohe Kohäsion, da in einer kleinen Einheit nur wenig Funktionalität, die dann aber stark zusammenhängt, implementiert werden kann.

Sowohl die Modulanzahl als auch die Einheitengröße werden für die Modularität-Metrik verwendet.

Wiederverwendbarkeit

Die Wiederverwendbarkeit lässt sich nicht direkt messen, allerdings kann gemessen werden wie oft Softwarekomponenten tatsächlich wiederverwendet werden, also die Wiederverwendung. Bei hoher Wiederverwendung muss auch die Wiederverwendbarkeit

hoch sein. Eine niedrige Wiederverwendung ist ein starker Indikator für niedrige Wiederverwendbarkeit.

Eine weitere Metrik für die Wiederverwendung ist die Code-Duplikation und die Einheitengröße. Wenn hohe Code-Duplikation vorhanden ist, ist die Wiederverwendung in der Regel gering, da duplizierter Code oft wiederverwendbarer implementiert werden kann. Bei geringer Code-Duplikation kann davon ausgegangen werden, dass eine hohe Wiederverwendung des Codes gegeben ist. Für eine hohe Wiederverwendbarkeit ist es wichtig, dass Einheiten nur wenige Aufgaben haben. Eine geringe Einheitengröße deutet daher auf eine höhere Wiederverwendbarkeit.

Analysierbarkeit

Nach [5] werden Code-Volumen, Code-Duplikation, Einheitengröße und Unit-Testabdeckung verwendet, um die Analysierbarkeit zu quantifizieren. Geringes Code-Volumen und geringe Einheitengröße hilft schnell einen Überblick über den Quellcode zu bekommen. Unit-Tests helfen die Funktionsweise des Quellcodes zu verstehen. Eine hohe Quellcode-Duplikation erschwert hingegen das Verständnis.

Modifizierbarkeit

In [5] wird die Komplexität der Einheiten und die Code-Duplikation als Metrik für die Modifizierbarkeit genannt. Code-Duplikation verringert die Modifizierbarkeit, da an Stellen mit dupliziertem Code, alle Teile des duplizierten Codes gleichermaßen geändert werden müssen. Das erhöht den Modifikations-Aufwand stark und erhöht die Chance, dass bei Modifikationen vergessen wird Teile des duplizierten Codes zu ändern und so Fehler im Programm entstehen. Eine hohe Komplexität der Einheiten sorgt dafür, dass es schwieriger ist diese Einheiten zu modifizieren, ohne Fehler in der Software zu erzeugen.

Testbarkeit

Die Testbarkeit von Software hängt nach [5] unter anderem von der Komplexität der Einheiten, der Einheitengröße und der Zahl der Unit-Testabdeckung ab. Eine hohe Unit-Testabdeckung ist ein Indikator für gute Testbarkeit, denn das bedeutet, dass die Software im allgemeinen testbar ist. Kleine Einheitengröße sorgt auch für eine bessere Testbarkeit, da es einfacher ist für kleine Einheiten, die weniger Aufgaben haben, Testfälle zu spezifizieren. Die Komplexität der Einheiten spielt auch eine Rolle, da für komplexe Einheiten voraussichtlich mehr und komplexere Testfälle spezifiziert werden müssen.

In Tabelle 1 wird die Zugehörigkeit der zuvor vorgestellten Quellcode Kennzahlen zu den Wartbarkeit-Subcharakteristika dargestellt. Diese Tabelle erweitert die in [5] vorgestellte Tabelle um die Änderungen die der neuere ISO 25010 Qualitätsstandard gegenüber dem alten ISO 9126 Standard bringt. Damit kann nun Software verglichen werden und Qualitätsänderungen durch Änderungen an der Software gemessen werden.

	Quellcodevolumen	Komplexität der Einheiten	Quellcode Duplikation	Einheitengröße	Unit-Testabdeckung	Modulanzahl	Wiederverwendung von Einheiten
Modularität				x		x	
Wiederverwendbarkeit			x	x			x
Analysierbarkeit	x		x	x	x		
Modifizierbarkeit		x	x				
Testbarkeit		x		x	x		

Tabelle 1: Zugehörigkeit der Kennzahlen zu den Wartbarkeit-Subcharakteristika

Im Gegensatz zu [5] wird hier keine konkrete Kennzahl für die Wartbarkeit vorgestellt. Vielmehr können mit dieser Metrik Softwarestände und Softwaresysteme relativ zueinander verglichen werden. Dafür werden die relativen Unterschiede in den einzelnen Metriken eins zu eins nach der Tabelle verrechnet, um einen Wert für die Jeweilige Subcharakterstik der Wartbarkeit zu erhalten. Um daraus einen Wert für die Wartbarkeit zu erhalten, werden die Werte der Subcharakteristika eins zu eins verrechnet.

Um diese Metrik zu verbessern, kann die Verrechnung der einzelnen Quellcode-Kennzahlen gewichtet werden. Diese Entscheidung kann je nach Anwendung der Metrik getroffen werden.

Es ist denkbar diese Metrik automatisch durch Softwaretools zu generieren. Das ermöglicht es diese Metrik, als Auswertung von Software in ein CI/CD System zu integrieren. Damit können Änderungen an der Software ausgewertet werden und gewarnt werden, wenn die Wartbarkeit durch Änderungen an der Software verschlechtert wird. Im extremsten Fall, kann eine Änderung auch abgelehnt werden, wenn die Wartbarkeit verschlechtert wird.

3.2 Sicherheit

Sicherheit ist ein Feld, welches in der Informatik sehr weit gefasst ist und in den meisten Themenbereichen eine elementare Rolle spielt. Daher ist es nicht verwunderlich, dass es schwer ist allgemeine Metriken für das Messen der Sicherheit von Software zu erarbeiten. Es gibt einige Artikel und Konferenz Papiere, die unterschiedliche Ansätze für das

definieren von Metriken verfolgen und es gibt einige Metaanalysen, die die Ergebnisse dieser Arbeiten miteinander vergleichen. Zusätzlich dazu wurden Standards definiert, welche jedoch weit gefasst sind und eine unpräzise Definition für Sicherheits-Metriken liefern [13].

Um einen konkreten Ansatz für die Bewertung der Sicherheit mittels Metriken zu finden werden drei unterschiedliche Ansätze (vereinfacht) vorgestellt [12] [3] [17] und miteinander Kombiniert. Ziel ist es so eine Sammlung an Metriken zu erhalten, welche eine Bewertung der Sicherheit einer Software über den Entwicklungsprozess hinweg ermöglichen. Abschließend wird bewertet, inwieweit die Sammlung der Metriken alle Subcharakteristika für Sicherheit im ISO 25010 Qualitätsmodell abdeckt.

3.2.1 Security Metrics and Software Development Progression [12]

In ihrem Artikel Security Metrics and Software Development Progression beschreiben die Autoren Smirity Jain und Maya Ingle eine Sammlung von Metriken, welche die Sicherheit von Software über den gesamten Entwicklungsprozesses hinweg messen. Mit diesen Metriken ist es möglich im nachhinein die Effektivität der fünf Phasen des Entwicklungsprozesses in Hinblick auf die Sicherheit zu messen. Im folgenden wird eine Auswahl der wichtigsten Metriken für die Phasen vorgestellt.

Phase 1: Anforderungsanalyse

Diese Phase konzentriert sich auf das Sammeln von (Sicherheits-) Anforderungen. Smirity und Ingle beschreiben hier vier Metriken für die Sicherheit.

- **Number of Security Requirements Gathered (NSRG)**: Anzahl der Sicherheitsanforderungen die in dieser Phase erfasst wurden.
- **Security Requirements Recorded Deviations (SRRD)**: Anzahl der Abweichungen von den zuvor definierten Sicherheitsanforderungen in den Anforderungsspezifikationen.
- **Security Requirements stage Security Errors (SRSE)**: Anzahl der Sicherheitsfehler, welche aus fehlerhaften Sicherheitsanforderungen resultieren.
- **Security Requirements Gathering Indicators (SRI)**: Misst den Einfluss der Analysephase auf die Anzahl der Sicherheitslücken.

Für die Auswertung dieser Metriken schlagen die Autoren vor, dass diese Metriken mit der Methode der kleinsten Quadrate zusammengefasst werden. Dabei werden die Metriken SRRD, SRSE und SRI als abhängig definiert, während NSRG als unabhängig betrachtet wird. Das Resultat dieser Rechnung ist die Funktion α, welche die Effektivität dieser Phase in Bezug auf die Sicherheit misst. Alpha wird mit der Formel 1 berechnet, wobei Y die Summe der Werte der Metriken SRRD, SRSE und SRI ist. αist im Interval von -1 bis 1 Aussagekräftig, ein größerer Wert bedeutet eine bessere Effektivität. Ein wert von -1 bedeutet das Sicherheit in der Phase nicht berücksichtigt wurde und ein

Wert größer als 1 bedeutet das die Effektivität nicht bewertet werden kann.

$$\alpha = \frac{(5,2421 - Y)}{NSRG} \tag{1}$$

Phase 2: Design

In dieser Phase werden die Anforderungen in ein Design überführt. Hier ist die Umsetzung von nicht-funktionalen Anforderungen, wie Fehlerbehandlung, Fehlermeldungen und Autorisierung wichtig. Die Autoren definieren für diese Phase drei Metriken.

- **Security Requirements Statistics (SRs)**: Prozentsatz der Sicherheitsanforderungen, welche in der Designphase bedacht wurden.
- **Design Tools und Test Effectiveness (DTTE)**: Misst den Anteil von Analysetools (z.B. Threat Analysis und Attack Patterns) und Testing an den Sicherheitsrelevanten Designaspekten.
- **Number of Design stage Security Errors (NDSE)**: Misst die Anzahl der Sicherheitsfehler, welche aus dem Softwaredesign resultieren.

Für die Auswertung dieser Metriken multipizieren die Autoren die Metriken DTTE und NDSE, das Ergebnis ist β. Da DTTE den Anzeil von Tools und Tests an dem resultierenden Design misst und NSDE die Fehler welche aus dieser Phase entstehen zählt, beschreibt βdie Effektivität der Designphase in Bezug auf die Sicherheit. βwird mit der Formel 2 berechnet und ist immer größer gleich 0. Der β-Wert sollte zwischen 0 und 1 liegen, ein größerer Wert bedeutet eine schlechtere Effektivität. Ein β-Wert von 0 impliziert, dass das Testing sehr Effektiv ist und es keine Fehler in der Designphase gibt.

$$\beta = NDSE * DTTE \tag{2}$$

Phase 3: Implementierung

In dieser Phase wird Sicherheit durch Input- und Outputvalidierung und durch die Verwendung von guten Programmierstandards erreicht. Die Autoren beschreiben hier drei Metriken um die Sicherheitsanstrengungen der Entwickler zu messen.

- **Percent of Secure Coding Aspects (PSCA)**: Misst das Verhälltnis der Sicherheitsrelevanten Coding-Aspekte in der Entwicklungsphase im Vergleich zu den Designaspekten (aus der Designphase).
- **Percent use of Coding Standards (PCS)**: Misst den Anteil an Coding-Standards für sichere Entwicklung, die verwendet wurden. Sie kann als ein Annährung für NSE dienen.
- **Number of Security Errors (NSE)**: Misst die Anzahl an Programmierfehlern, welche im Programm vorhanden sind oder durch externe Librarys eingebracht wurden.

Für die Auswertung verwenden die Autoren wird hier wieder die Methode der kleinsten Quadrate verwendet. Der resultierende Wert γ(Gleichung 3) beschreibt die Effektivität der Implementierungsphase in Bezug auf die Sicherheit. Ein negativer γWert deutet auf eine hohe Fehleranzahl hin und ein Wert nahe 0 deutet auf eine gute Effektivität der Phase hin.

$$\gamma = \frac{8,4022 - NSE}{PSCA + PCS} \tag{3}$$

Phase 4: Testen

In dieser Phase wird die Software auf Sicherheitslücken getestet. Die wichtigsten Metriken sind folgende.

- **Process Effectiveness (PE)**: Beschreibt das Verhältnis zwischen der Anzahl der entdeckten Sicherheitslücken und der Anzahl der Module, die Sicherheitstest unterzogen wurden.
- **Security Testing Ratio (STR)**: Misst den Anteil der Module, die Sicherheitstests unterzogen wurden.

Phase 5: Betrieb und Wartung

In dieser Phase werden Risikobewertungen und Sicherheitsanalysen durchgeführt um bei Patches eingebrachte Sicherheitsfehler zu finden. Außerdem findet hier das Logging, Monitoring und die Analyse von Sicherheitsereignissen statt. Die Autoren beschreiben hier X Metriken.

- **Mean Time to Complete Security Changes (MTCSC)**: Schätzt die durchschnittliche Zeit, die es braucht um einen Sicherheitsfehler zu beheben.
- **Rate of Vulnerability Assessments (RVA)**: Misst die Anzahl der Sicherheitsanalysen pro Quartal.
- **Ratio of Changes due to security considerations (RSC)**: Misst den Anteil der Änderungen am System auf Grund neuer Sicherheitsanforderungen im Vergleich zur gesammtheit der Systemänderungen.

3.2.2 Security Metrics for Source Code Structures [3]

In dem Paper Security Metrics for Source Code Structures werden drei Metriken vorgestellt, welche das Messen der Sicherheit auf Quellcode-Ebene ermöglicen. Ziel ist es eine Bewertung der Code-Qualität durchzuführen und festzustellen ob Fehler oder Unreinheiten vorhanden sind, die ein Angreifer ausnutzen könnte. Die Metriken sind folgende.

Stall Ratio (SR)

Die Stall Ratio misst, welcher Anteil des Programmcodes nicht benötigt wird, um dessen definiertes Ziel zu erreichen. Gerade in einer Schleife verzögern sogenannte Stall

Statements die Ausführung des Programms und können somit einen Angriff erleichtern. Die Stall Ratio wird mit der Formel 4 berechnet.

$$\text{Stall Ratio} = \frac{\text{Nicht-zielführende Zeilen in einer Schleife}}{\text{Gesamtanzahl der Zeilen in der Schleife}} \tag{4}$$

Nicht alle Stall Statements sind schlecht, gerade das Schreiben von Logs, Fehlermeldungen, etc. sind notwendig. Dennoch sollte auch die Anzahl sollten auch diese Stall Statements nicht allzu oft vorkommen. Angreifer nutzen häufig Denial of Service Angriffe um die Verfügbarkeit von Systemen zu beeinträchtigen. Eine hohe Anzahl an Stall Statements kann die Auswirkungen eines solchen Angriffs verstärken.

Coupling Corruption Propagation (CCP)

Kopplung von Methoden meint, dass zwei oder mehr Methoden voneinander abhängig sind. Ein Beispiel für eine Kopplung ist die Verwendung von globalen Variablen oder die Verwendung von Methoden aus anderen Klassen. Die Effekte einer Kopplung können sich schnell auf mehrere Methoden tiefer in der Kette auswirken. Angreifer können so Werte in einer Methode generieren, die dann Fehler in einer anderen Methode verursachen. Coupling Corruption Propagation (CCP) bestimmt die Anzahl der Methoden, die durch einen Fehler in einer Methode beeinflusst werden können. Hierfür wird die Anzahl der Kindmethoden gezählt, mit die einem Parameter aufgerufen werden, welcher auf einem Parameter der Originalmethode beruht. Mit einer geringeren Kopplung ist auch der CCP geringer. Kopplung ist häufig unumgänglich, Methoden mit einer hohen Kopplung sollten daher ausführlich getestet werden.

Critical Element Ratio (CER)

Bei objektorientierten Programmiersprachen gibt es Objekte, die während der Laufzeit des Programms initialisiert werden. Für diesen Prozess ist es aber nicht unbedingt nötig alle Aspekte eines Objekts zu initialisieren. Andere Aspekte des Objekts sind elementar für das Funktionieren des Programms, wenn sie verändert werden kann das Programm instabil werden und somit ein Sicherheitsrisiko darstellen. Es gilt, je höher die Anzahl dieser kritischen Elemente in einer Klasse, desto höher ist das Sicherheitsrisiko. Um dieses Risiko zu messen, wird der Anteil der kritischen Datenelemente in einem Objekt berechnet (CER). Wichtig ist hierbei, dass nur datenelemente Berücksichtigt werden, die während der Laufzeit des Programms durch einen Benutzer verändert werden können. Die Formel für die Berechnung des CER ist in der Formel 5 angegeben.

$$\text{Critical Element Ratio} = \frac{\text{Kritische Datenelemente in einem Objekt}}{\text{Gesamtanzahl der Elemente in einem Objekt}} \tag{5}$$

3.2.3 Security Metrics for Software Systems [17]

In dem Paper Security Metrics for Software Systems nutzen die Autoren CVEs (Common Vulnerabilities and Exposures) um Schwachstellen in einem Softwaresystem zu identifizieren. Die Menge an CVEs, für die ein System anfällig ist diehnt als Indikator für ein potentiell höheres Sicherheitsrisiko. Dabei wird auch die Schwere der CVEs nach CVSS, sowie die Wahrscheinlichkeit der Ausnutzung berücksichtigt.

Common Vulnerabilitie Scoring System (CVSS)

Das CVSS ist ein System zur Bewertung der Schwere und des damit einhergehenden Risikos einzelner Sicherheitslücken auf ein Computer-Umgbebungen. CVSS wurde von der US-Amerikanischen Organisation National Institute of Standards and Technology (NIST) entwickelt. Die Bewertung ist in 3 Kategorien unterteilt, die jeweils eine eigene Skala haben. Die Basismetrik misst die grundlegenden Eigenschaften einer Schwachstelle und ist Zeit- und Umgebungsunabhängig. Die zeitliche Metrik beschreibt die Schwere einer Schwachstelle in Abhängigkeit von der Zeit. Die Umgebungs-Metrik beschreibt die Schwere einer Schwachstelle in Abhängigkeit der Umgebung des Benutzers. Für die Bewertung werden mehrere Fragen zu den Metriken mit einem Wert zwischen 0 und 10 beantwortet. Der durchschnitt der Werte aller Fragen einer Metrik ergibt den Wert für dieser Metrik (0 - 10). Für die Zeitliche Bewertung wird die Bewertung der Basis- und der Zeitmetrik addiert. Für die Umgebungs-Bewertung werden alle drei Metriken miteinander verrechnet.

Common Vulnerabilities and Exposures (CVE)

Die Veröffentlichung von Sicherheitslücken findet in der Regel über die CVE-Datenbank statt, hier werden diese Lücken in einem einheitlichen Format veröffentlicht. Redhat schreibt dazu auf ihrer Webseite [14]:

> *CVE, kurz für Common Vulnerabilities and Exposures (Häufige Schwachstellen und Risiken), ist eine Liste mit öffentlichen Sicherheitsschwachstellen in Computersystemen. Mit CVE ist eine bestimmte Schwachstelle gemeint, der eine CVE-Nummer zugewiesen ist. In Sicherheitshinweisen von Anbietern und Forschenden wird fast immer mindestens eine CVE-Nummer erwähnt. Mithilfe von CVEs können IT-Fachleute solche Schwachstellen leichter priorisieren und beheben, um Computersysteme sicherer zu machen.*

Webseiten wie die NVD (National Vulnerability Database) bieten eine Schnittstelle zur CVE-Datenbank an und verknüpfen diese mit den entsprechenden Produkten und der CVSS Bewertung.

Software Sicherheits Metrik

Für das Erstellen der Metrik haben die Forscher nun vorgeschlagen, die wichtigsten Schwachstelle einer Software zu identifizieren und CVE einträge zu diesen zu finden. Für diese einzelne Schwachstellen wird dann die durchschnittliche CVSS Bewertung aller CVEs berechnet. So kann die Schwere der Schwachstellen gemessen werden. Um das Risiko (R) für ein Ausnutzen dieser Schwachstelle in die Bewertung einfließen zu lassen, schlagen die Forscher vor die Anzahl der CVEs einer Schwachstelle (K) mit dem Zeitraum (in Monaten), in dem die Schwachstelle in dem System existierte (M) zu verrechnen (siehe Gleichung 6).

$$R = \frac{K}{M} \tag{6}$$

	α-Wert	β-Wert	γ-Wert	PE und STR	MTCSC	RSC
Vertraulichkeit	x	x		x	x	x
Integrität	x		x	x	x	x
Unbestreitbarkeit	x		x			
Verantwortlichkeit	x	x	x			
Authentizität	x	x				

Tabelle 2: Zugehörigkeit der Metriken zu den Sicherheit-Subcharakteristika

Abschließend wird die durchschnittliche CVSS Bewertung aller Schwachstellen mit dem Missbrauchsrisiko verrechnet (siehe Gleichung 7). Mit dieser Formel lässt sich die Sicherheit der Software (s) berechnen.

$$SM\,(s) = \sum_{n=1}^{m} \left[\left(\frac{R_n}{\sum_{i=1}^{m} R_i} \right) \times W_n \right] \tag{7}$$

Verbesserungsansätze

In diesem Paper wenden die Autoren viel Zeit auf um Schwachstellen in dem System zu identifizieren und zu Kathegorisieren. Dieser händische Prozess lässt sich durch die Verwendung der National Vulnerability Database (NVD) deutlich vereinfachen. Hier werden Softwaresysteme und Packete aufgelistet, welche Schwachstellen aufweisen und es wird Angemerkt ab welcher Version diese Schwachstelle behoben wurde. Zur vereinfachung dieses Prozesses kann es Hilfreich sein einen Scanner zu verwenden, welcher die verwendete Software und Packete identifiziert und die Schwachstellen in der NVD Datenbank mit den identifizierten Packeten vergleicht. Die gefundenen Schwachstellen können zusammen mit dem Zeitraum, in dem der Scanner diese im System gefunden hat, verwendet werden um die Metrik zu berechnen. So kann eine höhere Automatisierung erreicht werden.

3.2.4 Kombination der Ansätze und Bewertung der Metriken

In Tabelle 2 und 3 ist aufgeschlüsselt welche Charakteristika der ISO 25010 durch die Metriken abgedeckt werden. Da in der Analysephase die Grundlagen für alle weiteren Phasen der Softwareentwicklung gelegt werden, deckt der α-Wert alle Charakteristika der ISO 25010 ab. Tabelle 2 gibt einen Überblick über die Abdeckung der Metriken des ersten Artikels [12] Der β-Wert gibt eine Aussage über die Effektivität der Designphase, der Fokus liegt daher auf der Identifikation, Authorisierung und weiteren

	SR	CCP	CER	CVE-Metrik
Vertraulichkeit				x
Integrität				x
Verantwortlichkeit	x	x	x	x
Modifizierbarkeit				x
Authentizität				x

Tabelle 3: Zugehörigkeit der Metriken zu den Sicherheit-Subcharakteristika

nicht-funktionalen Anforderungen. Der γ-Wert gibt eine Aussage über die Effektivität der Implementierungsphase, daher ist er wichtig für die Integrität, Unbestreitbarkeit und Verantwortlichkeit. Process Effectiveness und Security Testing Ratio ermöglichen die Bewertung der Testphase und die Effektivität darin Fehler und Sicherheitslücken festzustellen. Daher liegt der Fokus hier auf Vertraulichkeit und Integrität. Dasselbe gilt auch für die Metriken der Betriebs- und Wartungsphase.

Tabelle 3 gibt einen Überblick über die Abdeckung der Metriken der anderen beiden Paper [3] [17]. Da das erste Paper sich auf die Sicherheit auf der Quellcode-Ebenen konzentriert, deckten die Metriken nur das Charakteristika der Integrität ab. Die Tiefe der Analyse ist macht diese Metriken sehr effektiv für die Analyse des Quellcodes. Das zweite Paper Fokussiert sich auf die Bewertung von Softwaresystemen auf Basis der Anfälligkeit für bekannte Schwachstellen. Da die Art der Schwachstellen vielseitig ist können so abhängig von den betrachteten Systemen und Komponenten auch alle Charakteristika der ISO 25010 abgedeckt werden.

Zusammengenommen bieten die Metriken der Paper eine gute Abdeckung der Charakteristika der ISO 25010 und ermöglicen die Bewertung sowohl auf der tiefen Quellcode-Ebene, als auch mit Hinblick auf das System als ganzes. Zusätzlich wird auch der Prozess der Sofwareentwicklung berücksichtigt, der eine elementare Rolle in der Entwicklung von guter, sicherer Software spielt.

4 Ausblick

Die hier vorgestellte Metrik zur Wartbarkeit macht es zwar einfach die Wartbarkeit von Softwareprodukten und Softwaresystemen zu messen, allerdings hat diese Metrik auch einige Nachteile. So kann diese Metrik keine direkte Aussage über die Dauer von Wartungsaufgaben an der Software machen. Außerdem wurde nicht verglichen wie sich diese Metrik im Vergleich mit anderen Wartbarkeit-Metriken verhält. Es ist denkbar, dass die hier vorgestellte Metrik völlig andere Werte als andere Wartbarkeit-Metriken liefert. In einer weiteren Arbeit könnte darauf eingegangen werden und die hier vorgestellte Wartbarkeit-Metrik mit andere Wartbarkeit-Metriken verglichen werden, um

festzustellen, inwieweit sich diese unterscheiden.

Außerdem bietet die hier vorgestellte Wartbarkeit-Metrik die Möglichkeit Gewichte zu setzen, um die Metrik weiter zu verbessern und anzupassen. In einer weiteren Arbeit könnte diese Metrik durch die Gewichtungen an andere Wartbarkeit-Metriken angepasst werden.

In einer weiteren Arbeit ist es außerdem Denkbar weitere Metriken zu finden, mit denen die übrigen Qualitätscharakteristika einfach quantifiziert werden können.

Literatur

[1] Ben Boyter. *Sloc Cloc and Code (scc)*. https://github.com/boyter/scc. 2022.

[2] BSI. *Die Lage der IT-Sicherheit in Deutschland 2022*. de. Die Lage der IT-Sicherheit in Deutschland BSI-LB22/511. Bonn, Okt. 2022, S. 115. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2022.html.

[3] Istehad Chowdhury, Brian Chan und Mohammad Zulkernine. „Security metrics for source code structures". en. In: *Proceedings of the fourth international workshop on Software engineering for secure systems*. Leipzig Germany: ACM, Mai 2008, S. 57–64. ISBN: 9781605580425. DOI: 10.1145/1370905.1370913. URL: https://dl.acm.org/doi/10.1145/1370905.1370913.

[4] Statista Research Department. *Beschwerden über Internetkriminalität in den USA 2021*. de. Dez. 2022. URL: https://de.statista.com/statistik/daten/studie/154433/umfrage/beschwerden-ueber-internetkriminalitaet-seit-2000/.

[5] Ilja Heitlager, Tobias Kuipers und Joost Visser. „A Practical Model for Measuring Maintainability". In: *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*. IEEE. 2007, S. 30–39. DOI: 10.1109/QUATIC.2007.8.

[6] „IEEE Standard for Information Technology–System and Software Life Cycle Processes–Reuse Processes". In: *IEEE Std 1517-2010 (Revision of IEEE Std 1517-1999)* (2010), S. 1–51. DOI: 10.1109/IEEESTD.2010.5551093.

[7] ISO 25010. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. Standard. Geneva, CH: International Organization for Standardization, 2011.

[8] ISO 27000. *Information technology — Security techniques — Information security management systems — Overview and vocabulary*. Standard. Geneva, CH: International Organization for Standardization, 2018.

[9] ISO 7498-2. *Information processing systems -Open Systems lnterconnection -Basic Reference Model. Part 2: Security Architecture*. Standard. Geneva, CH: International Organization for Standardization, 1989.

[10] ISO 9126-1. *Information technology — Software product quality. Part 1: Quality model*. Standard. Geneva, CH: International Organization for Standardization, 2000.

[11] „ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary". In: *ISO/IEC/IEEE 24765:2017(E)* (2017), S. 1–541. DOI: 10.1109/IEEESTD.2017.8016712.

[12] Smriti Jain und Maya Ingle. „Security Metrics and Software Development Progression". In: 2014.

[13] Daniel Mellado, Eduardo Fernández-Medina und Mario Piattini. „A comparison of software design security metrics“. en. In: *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*. Copenhagen Denmark: ACM, Aug. 2010, S. 236–242. ISBN: 9781450301794. DOI: 10.1145/1842752.1842797. URL: https://dl.acm.org/doi/10.1145/1842752.1842797.

[14] RedHat. *Was bedeutet CVE?* de. Nov. 2021. URL: https://www.redhat.com/de/topics/security/what-is-cve.

[15] Muhammad Danish Roshaidie u. a. „Importance of Secure Software Development Processes and Tools for Developers“. In: (2020). DOI: 10.48550/ARXIV.2012.15153. URL: https://arxiv.org/abs/2012.15153.

[16] Mohammad Tahaei u. a. „“I Don’t Know Too Much About It”: On the Security Mindsets of Computer Science Students“. en. In: *Socio-Technical Aspects in Security and Trust*. Hrsg. von Thomas Groß und Theo Tryfonas. Bd. 11739. Cham: Springer International Publishing, 2021, S. 27–46. ISBN: 9783030559571 9783030559588. DOI: 10.1007/978-3-030-55958-8_2. URL: https://link.springer.com/10.1007/978-3-030-55958-8_2.

[17] Ju An Wang u. a. „Security metrics for software systems“. en. In: *Proceedings of the 47th Annual Southeast Regional Conference*. Clemson South Carolina: ACM, März 2009, S. 1–6. ISBN: 9781605584218. DOI: 10.1145/1566445.1566509. URL: https://dl.acm.org/doi/10.1145/1566445.1566509.