FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Software-Produktmetriken zur Erstellung eines Software-Qualitätsmodells

Projektarbeit

Friedrich-Schiller Universität Jena
Fakultät für Mathematik und Informatik

eingereicht von

Celestino Madera Castro
Matrikelnummer: 153138
geboren am 17.11.1996 in Jena

15. Dezember 2022

Inhaltsverzeichnis

1 Einleitung

Erfolgreiche Sicherheitsattacken auf Software-Anwendungen können zu potenziellen Gefahren für deren Benutzern und Eigentümern führen. Zur Ausübung solcher bösartigen Zwecke werden in der Regel Schwachstellen eines Systems ausgenutzt. Schwachstellen beruhen auf Fehlern und Bugs, welche beispielsweise durch Exploits ausgenutzt werden können und Zugriff zu vertraulichen Informationen ermöglichen oder diese Daten manipulieren. Dabei bildet ein Fehler, welcher beispielsweise innerhalb einer Spezifikation, der Entwicklung oder der Konfiguration einer Anwendung auftreten kann, einen Schwachpunkt als Schnittstelle für Angreifer.

Software-Produktmetriken dienen zur Quantifizierung der Eigenschaften einer Software in allen Phasen des Entwicklungsprozesses. Solche Metriken stellen ein essentielles Hilfsmittel zur Messung solcher Anwendungen dar und ermöglichen die Erfassung der Eigenschaften von Quellcode-Abschnitten, welche Schwachstellen enthalten. Durch den Vergleich solcher Abschnitte mit anderen Quellcode-Abschnitten können wiederum weitere Schwachstellen innerhalb einer Software-Anwendung erkannt werden. Durch eine frühzeitige Erkennung solcher Schwachstellen können wiederum verfügbare Ressourcen effizient verteilt werden, um die zugrunde liegenden Fehler zu behen und letztendlich die Qualität der Software zu erhöhen.

Neben der Erkennung von Fehlern und Schwachstellen ermöglichen Software-Produktmetriken somit ebenfalls die Bewertung der Qualität einer Software-Anwendung. Dabei beruht die Bewertung der Software-Qualität auf dem Einsatz eines ausgewählten Modells unter Verwendung verschiedener Software-Produktmetriken. Aufgrund der stetigen Forschung werden regelmäßig Produktmetriken ohne Validierung auf der Grundlage von empirischen Studien veröffentlicht. Um ein aussagekräftiges Modell zu entwickeln werden jedoch Metriken benötigt, welche auf einer zuverlässigen wissenschaftlichen Grundlage basieren.

Im Rahmen dieser Projektarbeit soll ein katalog-artiger Überblick von Software-Metriken erstellt werden. Dabei sollen die in diesem Überblick enthaltenen Metriken alle möglichen Aspekte eines Software-Produkts abdecken, um das vollständige Potenzial der Software-Messung zu gewährleisten. Auf Grundlage eines solchen Überblick lassen sich wiederum durch fachliche Informationen Metriken auswählen, welche sich zur Erkennung von Fehlern und der Bewertung von Software-Qualität eignen.

Deshalb wird im Rahmen dieser Hausarbeit eine Taxonomie von Software-Produktmetriken präsentiert und Hinweise bezüglich der wissenschaftlichen Verwendung verschiedener Metriken bereitgestellt. In Kapitel 2 werden die grundlegenden Prozesse zur Erstellung der Metriken-Taxonomie formuliert, indem Forschungsfragen definiert worden und die verwendete Literatur charakterisiert wird. Die letztendliche Metriken-Taxonomie wird in Kapitel 3 präsentiert. In diesem Kapitel werden außerdem weitere Informationen bezüglich der Verwendung von Software-Produktmetriken beschrieben. In Kapitel 4 werden letztendlich die Metriken-Taxonomie inklusive ihres Erstellungsprozesses rückblickend zusammengefasst und die Forschungsfragen beantwortet, ehe letztendlich ein Ausblick hinsichtlich weiterer Schritte zur Erstellung eines Software-Qualitätsmodells gegeben wird.

2 Wissenschaftliches Vorgehen und Methodik

Die im Rahmen dieser Hausarbeit erstellte Taxonomie soll einen Überblick der zur Erstellung eines Software-Qualitätsmodells geeigneten Software-Metriken gewährleisten. In Kapitel 2.1 werden dafür zuerst Forschungsfragen beschrieben, welche mithilfe der konstruierten Taxonomie beantwortet werden sollen. Anschließend wird in Kapitel 2.2 der wissenschaftliche Prozess zur Erstellung einer solchen Taxonomie beschrieben und erklärt, wie dadurch die Forschungsfragen beantwortet werden können. In Kapitel 2.3 wird letztendlich die verwendete Literatur thematisiert, indem die jeweiligen Problemstellungen und Herangehensweisen der einzelnen Forschungsarbeiten betrachtet werden.

2.1 Forschungsfragen

Durch die Konstruktion der Metriken-Taxonomie sollen verschiedene Forschungsfragen beantwortet werden. Das Ziel dieser Fragestellungen ist es, einen umfangreichen Überblick über alle möglichen Arten von Software-Metriken zu gewährleisten. Darüber hinaus soll überprüft werden, welche Metriken bereits intensiver betrachtet wurden und häufig in verschiedenen Modellen eingesetzt wurden. Dementsprechend wurden im Rahmen dieser Hausarbeit die folgenden Forschungsfragen aufgestellt:

- **RQ1**: Welche Software-Metriken existieren und wie sind diese Metriken definiert?
- **RQ2**: Welche verschiedenen Arten von Software-Metriken existieren?
- **RQ3**: Welche Software-Metriken wurden zur Erstellung von Software-Qualitätsmodellen bereits untersucht?

Zur Beantwortung der ersten Forschungsfrage RQ1 sollen Software-Metriken umfrangreich ermittelt werden. Dazu sollen die Bezeichnungen, Abkürzungen und Definitionen einzelner Metriken tabellarisch aufgelistet werden. Um die zweite Forschungsfrage RQ2 zu beantworten, sollen alle untersuchten Metriken kategorisiert werden. Die Erstellung einer Metriken-Taxonomie eignet sich dabei als effektiver Lösungsansatz zur Identifikation der verschiedenen Arten von Metriken. Zur Beantwortung der dritten Forschungsfrage RQ3 sollen zusätzliche Informationen während der Ermittlung von Software-Metriken und der Erstellung der Metriken-Kategorien dokumentiert werden, welche sich auf den Einsatz von Software-Metriken in verschiedenen Anwendungsgebieten beziehen. Die Verwendung von Metriken zur Bewertung der Software-Qualität soll dabei explizit hervorgehoben werden.

2.2 Vorgehen

Im Rahmen einer Literaturrecherche sollen alle benötigten Informationen zur Beantwortung der Forschungsfragen ermittelt werden. Durch die Anwendung des Schneeballsystems soll ausgewählte Literatur initial analysiert werden. Die Betrachtung referenzierter Literatur soll dabei unterstützen, zusätzliche oder fehlende Informationen zu identifizieren, um die Forschungsfragen umfassend beantworten zu können. Die initial untersuchten Studien werden in Kapitel 2.3 betrachtet.

Um alle benötigten Informationen zu erhalten, wurden alle verwendeten Metriken aus den initial untersuchten Studien ermittelt. Die Bezeichnung dieser Metriken inklusive verwendeter Abkürzungen wurden anschließend aufgelistet. Um einen Überblick über diese Metriken bereits

zu Beginn der Forschungsarbeit zu gewährleisten, wurden diese Metriken grob strukturiert. Diese Strukturierung war nach initialer Untersuchung der verschiedenen Studien bereits möglich und diente als Ausgangslage zur Erstellung der Metriken-Taxonomie. Dementsprechend wurden alle ermittelten Metriken in entsprechende Kategorien unterteilt.

Auf Grundlage dieser Strukturierung wurden für sämtliche Metriken deren Definitionen oder Berechnungsformeln ermittelt, um die Korrektheit der Grobstrukturierung zu überprüfen. Falls zu wenige Informationen bezüglich einzelner Metriken in der initial verwendeten Literatur verfügbar war, wurde die referenzierte Literatur untersucht, um entsprechende Informationen bereitzustellen. Gegebenenfalls wurden Software-Metriken dieser Studien der Grobstrukturierung hinzugefügt, falls diese Metriken nicht bereits erfasst worden. Ermittelte Software-Metriken für die nach dieser Recherche zu wenige Informationen ermittelt worden, wurden dabei aus der Auflistung entfernt.

Die erstellte Metriken-Liste wurde anschließend anhand der ermittelten Definitionen auf Duplikate von Metriken überprüft. Diese Duplikate entstanden durch die Zusammenführung von Metriken verschiedener Quellen, welche teilweise für verschiedene Metriken eine unterschiedliche Namensgebung verwendeten. Solche Duplikate verfügen meistens über unterschiedliche Bezeichnungen und Abkürzungen. Sie bilden jedoch in einem identischen Kontext die selben Informationen ab. Da der Überblick aller Software-Metriken redundant sein soll, wurden manuell ermittelte Duplikate vereinigt. Dazu wurde die Namensgebung und Liste möglicher Abkürzungen entsprechend angepasst. Darüber hinaus wurden die Abkürzungen aller Metriken angepasst, sodass jede Metriken-Abkürzung genau eine Metrik repräsentiert.

Nachdem die Korrektheit des grob strukturierten Metriken-Überblicks sichergestellt worde, wurde jede einzelne Metriken-Kategorie betrachtet. Innerhalb jeder Kategorie wurden alle Metriken miteinander gruppiert, welche Informationen aus einem ähnlichen Kontext beziehen oder über ähnliche Charakteristiken und Eigenschaften verfügen. Anhand dieser Gruppierungen wurden Unterkategorien definiert, welche den Metriken-Überblick verfeinern und letztendlich die erstellte Taxonomie repräsentieren.

Mithilfe der Taxonomie werden letztendlich die Forschungsfragen RQ1 und RQ2 beantwortet, da sie alle ermittelten Software-Metriken sowie alle definierten Metriken-Arten enthält. Die Forschungsfrage RQ3 wird hingegen durch die während der Literaturrecherche gesammelten Informationen explizit beantwortet.

2.3 Verwendete Literatur

Als Einstiegspunkt der Literaturrecherche wurden zwei verschiedene wissenschaftliche Forschungsartikel ausgewählt. Der erste Artikel beschreibt eine systematische Mapping-Studie von Quellcode-Metriken und wurde im Jahr 2017 veröffentlicht [1]. Sie dient als zentrales Element der Literaturrecherche, da innerhalb dieser Studie bereits eine umfangreiche Menge an Software-Metriken untersucht wurde. In Kapitel 2.3.1 wird die darin verwendete Methodik zur Erfassung der Metriken beschrieben.

Die zweite Quelle basiert auf einer Vergleichsstudie projektübergreifender Ansätze zur Fehlervorhersage, welche im Jahr 2018 veröffentlicht wurde [2]. Diese Studie beschreibt ein Modell zur Bewertung solcher Ansätze auf der Grundlage von Datensätzen verschiedener Software-Produkte. Dazu werden in Kapitel 2.3.2 der wissenschaftliche Prozess sowie die Charakteristiken der verwendeten Datensätze beschrieben.

2.3.1 Systematische Mapping-Studie von Quellcode-Metriken

Das Ziel dieser Studie war es, den aktuellen Forschungsstand bezüglich der Verwendung von Quellcode-Metriken basierend auf empirischen Studien zu ermitteln. Es wurden 226 verschiedene Studien untersucht, welche in einem Zeitraum von 2010 bis 2015 veröffentlicht wurden. Innerhalb dieser Studien wurden über 300 verschiedene Quellcode-Metriken der vier aktuell messbaren Programmierparadigmen erfasst. Zusätzlich wurden Verfahren und Werkzeuge zur Bereitstellung und Extraktion von Quellcode-Metriken sowie Arten gemessener Systeme identifiziert.

Auf Grundlage dieser Informationen soll diese sekundäre Studie durch die Beantwortung aufgestellter Forschungsfragen als Überblick des aktuellen Forschungsstandes bezüglich Quellcode-Metriken dienen. Aufgrund der jährlichen Publikation von Forschungsarbeiten und der hohen Vielfalt an Metriken aus aktueller Literatur wurden zur Beschreibung des aktuellen Forschungsstand entsprechende Beweise aus den einzelnen Studien gesammelt. So konnten durch die Planung, Verwaltung und Dokumentation dieses sekundären Review-Prozesses umfassende Aspekte des gesamten Themenumfelds betrachtet werden, um beispielsweise aktuelle Trends von Quellcode-Metriken zu betrachten. Dabei diente die Review-Planung zur Ermittlung des Bedarfs, der Definition von Forschungsfragen und zur Erstellung eines Review-Protokolls.

Für die Auswahl zu untersuchender Studien wurde ein Suchdesign definiert. Im Rahmen dieses Designs wurde eine Menge elektronischer Datenbanken bestimmt, auf denen die systematische Suche ausgeführt wurde. Dazu wurden die elektronischen Datenbanken *IEEEXplore*, *Elsevier's Science Direct* und *ACM Digital Library* ausgewählt. Zusätzlich wurde ein Zeitraum definiert, in dem Forschungsarbeiten innerhalb der elektronischen Datenbank veröffentlicht worden. Auf Grundlage dieser Kriterien wurde unter Verwendung der folgenden Suchmuster automatische Suchen durchgeführt:

- source code metrics
- object-oriented metrics
- aspect-oriented metrics
- feature-oriented metrics
- software metrics
- object oriented metrics
- aspect oriented metrics
- feature oriented metrics

Nach Ausführung dieser systematischen Suche wurden weitere Studien mittels manueller Suche durch Anwendung des Schnellballprinzips ermittelt. Anschließend wurden alle erfassten Studien anhand bestimmter Kriterien zur Inklusion und Exklusion überprüft. Darüber hinaus wurde die Qualität dieser Studien anhand vier weiterer Kriterien bewertet, um die Aussagekraft dieser Studien repräsentieren.

Zur Bereitstellung eines Überblicks erfasster Quellcode-Metriken wurden relevante Daten aus den einzelnen Studien extrahiert. Dazu wurden verschiedene Zählvariablien definiert, welche über einen Zusammenhang mit den zur Beantwortung der Forschungsfragen benötigten Daten verfügen. Auf Grundlage dieser Zählungen wurden anschließend statistische Analysen durchgeführt. Dafür wurden das Veröffentlichungsjahr, der Veröffentlicher, die Veröffentlichungsart und der Name des Events oder des Journals einzelner Studien erfasst. Weiterhin wurden darin verwendete Programmierparadigmen und Quellcode-Metriken sowie die Themen und Anwendungsfälle dieser Komponenten ermittelt. Darüber hinaus wurden neue, vorgeschlagene Metriken identifiziert und die Namen von Tools zur automatisierten Extraktion von Metriken-Werten sowie verwendete Datensätze

für den Erhalt von Metriken-Werten ermittelt. Der Name gemessener Systeme sowie die darin verwendeten Programmiersprachen sowie zusätzliche Eigenschaften dieser Systeme wurden ebenfalls erfasst.

2.3.2 Vergleichsstudie zur Bewertung projektübergreifender Ansätze der Fehlervorhersage

Die Vorhersage von Defekten unter Verwendung von Trainingsdaten anderer Projekte wird als projektübergreifende Fehlervorhersage bezeichnet. Sie stellt ein großes Unterthema der Software-Defekt-Vorhersage seit dem letzten Jahrzehnt dar und dient der Qualitätssicherng von Software-Projekten. Aufgrund des Mangels an Replikationsmöglichkeiten von Ergebnissen ist der aktuelle Stand der Forschung sehr unklar definiert, sodass eine optimale Vorhersage nicht gewährleistet werden kann. Verschiedene Experimente verfügen meistens über unterschiedliche Konfigurationen. So werden beispielsweise verschiedene Performanz-Metriken basierend auf unterschiedlichen zugrunde liegenden Daten berechnet. Dadurch können verschiedene Ansätze schwer miteinander verglichen werden, wodurch die Auswahl eines optimalen Ansatzes nicht möglich ist.

Im Rahmen einer Forschungsarbeit wurde ebenfalls eine Mapping-Studie durchgeführt, in der Studien bezüglich der Aspekte projektübergreifender Fehlervorhersage untersucht worden. Dazu wurden 50 zwischen 2002 und 2015 veröffentlichte Publikationen untersucht, welche als Grundlage dieser Vergleichsstudie dienen. Mithilfe dieser Mapping-Studie wurden wiederum 24 durch Forscher vorgeschlagene Ansätze der strikten projektübergreifenden Fehlervorhersage repliziert, welche zwischen 2008 und 2015 veröffentlicht worden. Für den Vergleich dieser Ansätze wurde letztendlich ein Benchmark erstellt, welcher fünf verschiedene Datensätze verwendet. Diese Datensätze sind öffentlich verfügbar und werden in Tabelle 1 kurz beschrieben.

Name	Datensatz	Metriken
JURECZKO	Daten aus insgesamt 92 verschiedenen Versionen von Software-Produkten bestehend aus 48 Versionen 15 verschiedener Open-Source-Projekte, 27 Versionen privater Projekte und 17 Versionen verschiedener von Studenten implementierter akademischer Projekte	20 statische Produkt-Metriken für Java-Klassen inklusive der Anzahl gefundener Defekte pro Klasse
MDP	Vorverarbeitete Version des von NASA veröffentlichten Datensatzes *Metrics Data Programm* bestehend aus 12 Produkt-Versionen aus 6 Projekten	17 statische Quellcode-Metriken
AEEEM	Versionen von 5 Java-Produkten verschiedener Software-Projekte	71 Software-Metriken bestehend aus Prozess-Metriken und statischen Produkt-Metriken
NETGENE	Daten aus 4 Open-Source-Projekten mit Verfolgung strikter, industriebezogener Entwicklungsprozesse	465 Metriken inklusive statischer Produkt-Metriken, Netzwerk-Metriken und Stammbaum-Metriken
RELINK	Defekt-Informationen von 3 Produkten verschiedener Projekte	60 statische Produkt-Metriken und 3 unterschiedlichen Defekt-Bezeichnungen für jedes Modul

Tabelle 1: Überblick der in [2] verwendeten Datensätze

Durch den Ansatz der strikten projektübergreifenden Fehlervorhersage wird aus jedem Datensatz ein Software-Produkt als Zielprodukt ausgewählt. Die Informationen aller weiteren Software-Produkte der jeweiligen Datensätze werden anschließend als Trainingsdaten für das Modell der Fehlervorhersage verwendet. Alle weiteren Revisionen des Zielprodukts werden hingegen nicht weiter berücksichtigt. Zur Auswertung der Performanz der einzelnen Ansätze auf verschiedenen Software-Produkten wurden die Performanz-Metriken Recall, Precision, Accuracy, F-Measure, G-Measure, MCC und AUC verwendet. Dazu wurde ein systematischer Vergleich zur Platzierung aller 24 verschiedenen Ansätze durchgeführt, indem die Kombinationen der unterschiedlichen Performanz-Metriken analysiert worden. Die dafür verwendeten Techniken und die daraus resultierenden Ergebnisse sind wiederum vollständig verfügbar und als Open-Source-Werkzeug verfügbar.

Auf Grundlage dieser Experimente sollten die Ansätze zur projektübergreifenden Fehlervorhersage identifiziert werden, welche hinsichtlich der einzelnen Performanz-Metriken die beste Performanz aufweisen. Ansätze mit konsistenter Erfüllung von Performanz-Kriterien sollten ebenfalls ermittelt werden. Darüber hinaus sollte der Einfluss auf die Ergebnisse des Benchmarks hinsichtlich der Verwendung größerer Software-Produkte überprüft werden. Der Einfluss auf die Benchmark-Ergebnisse sollte außerdem bezüglich der Verwendung relativ kleiner Teilmengen von Daten betrachtet werden.

3 Software-Produktmetriken

Software-Produktmetriken erfassen die Eigenschaften von Software-Produkten durch die Abbildung verschiedener Charakteristiken gemessener Entitäten. Diese Charakteristiken werden in der Regel als numerische Werte dargestellt. In besonderen Fällen können sie auch als ordinäre oder kategorische Werte repräsentiert werden. Zur Berechnung dieser Metriken werden Informationen aus dem Quellcode extrahiert und gegebenenfalls weiterverarbeitet. Dabei können die Werte einzelner Metriken auch auf Grundlage bereits berechneter Metriken ermittelt werden.

Durch die Auswertung dieser Metriken können letztendlich Aussagen über die Qualität eines Software-Artefakts getroffen werden. Da die Bewertung der Software-Qualität Bestandteil eines Software-Produkts ist, stellen Software-Metriken wiederum eine essentielle Komponente im Software-Messeungsprozess dar. Während jeder Phase des Software-Entwicklungsprozesses wird dieser Prozess benötigt, um die Qualität einer Software zu überwachen. Dazu werden während der Entwicklung verschiedene Metriken vieler Zwischenprodukte gemessen, ehe eine lieferbare Software über eine angemessene Qualität verfügt.

Eine Übersicht aller verwendbaren Arten von Software-Produktmetriken wird in Kapitel 3.1 präsentiert. In Kapitel 3.2 werden zu diesen Metriken zusätzliche Informationen bezüglich ihrer Verwendung beschrieben.

3.1 Metriken-Taxonomie

Auf Grundlage der beschriebenen Methodik wurden 344 verschiedene Software-Produktmetriken aus der initial untersuchten Literatur und den darin referenzierten Quellen erfasst. Viele dieser Metriken können hinsichtlich der Programmierparadigmen kategorisiert werden, auf denen sie grundsätzlich konzipiert worden. Ein solches Paradigma repräsentiert einen Programmierstil, welches verschiedene Prinzipien und Herangehensweisen zur Entwicklung einer Software umfasst. Im Rahmen dieser Arbeit wird zwischen den aktuell vorherrschenden Paradigmen der objekt-orientien, aspekt-orientierten, merkmals-orientierten und prozeduralen Programmierung unterschieden. Darüber hinaus existieren weitere Paradigmen, welche jedoch in der Regel seltener eingesetzt werden und daher für weitere Betrachtungen nicht berücksichtigt worden.

Die prozedurale Programmierung war das erste dominante Paradigma der Software-Entwicklung, weshalb ursprünglich die meisten Produktmetriken dafür konzipiert worden. Seit 1990 ist die objekt-orientierte Programmierung das am meisten untersuchte Programmierparadigma. Bis heute wurde dieses Paradigma am häufigsten in Software-Projekten eingesetzt und im Rahmen von Forschungsarbeiten analysiert. Dementsprechend wurden die meisten erfassten Software-Metriken auf der Basis objekt-orientierter Prinzipien entwickelt. Viele Metriken der prozeduralen Programmierung können außerdem für dieses Paradigma ohne weitere Anpassungen ebenfalls verwendet werden.

Die Paradigmen der aspekt- und merkmals-orientierten Programmierung wurden hingegen wegen der Dominanz objekt-orientierter Programmierung weniger erforscht. Beide Paradigmen können als spezialisierte Erweiterung der objekt-orientierten Programmierung berachtet werden und verfügen über besondere Vorteile hinsichtlich der Programmierung und der Produktlinien einer Software. Dementsprechend können einige Metriken der objekt-orientierten Programmierung durch grundlegende Anpassungen ebenfalls für diese Programmierparadigmen verwendet werden. Erst in den letzten beiden Jahrzehnten stieg jedoch das Interesse an diesen Paradigmen aufgrund

der Veränderungen von Anforderungen und Prozessen der Software-Entwicklung. Dadurch werden vor allem für die merkmals-orientierte Programmierung weitere Studien zur Definition angemessener Software-Metriken benötigt.

Eine initiale Kategorisierung aller erfassten Metriken hinsichtlich ihrer zugrunde liegenden Programmierparadigmen eignet sich jedoch nur hinreichend. Die meisten Metriken wurden für die objekt-orientierte Programmierung konzipiert und können durch spezifische Anpassungen ebenfalls in Software-Projekten eingesetzt werden, welche andere Programmierparadigmen verwenden. Somit würde eine solche Einteilung keine essenziellen Eigenschaften einzelner Software-Metriken berücksichtigt und gleichzeitig hervorheben. Darüber hinaus existieren Software-Metriken, welche allgemeingültig definiert worden und in jedem Programmierstil verwendet werden können. Eine Einteilung aller Software-Metriken anhand fundamentaler Eigenschaften ist daher empfehlenswerter.

Zur Einteilung aller Software-Metriken wurden dafür die folgenden Kategorien konstruiert:

- Quantifizierungsmetriken
- Komplexitätsmetriken
- Kohäsionsmetriken
- Kopplungsmetriken
- Vererbungsmetriken
- Sicherheitsmetriken
- Netzwerkmetriken

Aufgrund der hohen Anzahl erfasster Metriken wurde diese Kategorisierung zur Optimierung des Überblicks verfeinert. Dazu eignet sich unter anderen die Differenzierung zwischen den Metriken verschiedener Programmierparadigmen. Bei Existenz aspekt- oder merkmals-orientierter Metriken innerhalb einer Metriken-Kategorie wurden entsprechende Unterkategorien integriert. Zwischen Metriken prozeduraler und objekt-orientierter Programmierung wurde hingegen nicht differenziert, da die Metriken prozeduraler Programmierung ebenfalls für Software-Projekte objekt-orientierter anwendbar sind. Da die Menge dieser Metriken jedoch relativ hoch ist, wurde die Strukturierung dieser Metriken aufgrund weiterer essenzieller Eigenschaften verfeinert.

3.1.1 Quantifizierungsmetriken

Innerhalb dieser Metriken-Kategorie werden alle Software-Metriken zusammengefasst, welche das Auftreten bestimmter Elemente zählen. Auf Grundlage dieser Zählungen finden in der Regel keine weiteren Berechnungen und Aggregationen zur Ermittlung von Metriken-Werten statt. Somit wird beim Zählen der Vorkommnisse von Elementen auch keine Gewichtungen berücksichtigt und es werden keine Zählungen in das Verhältnis zu jeglichen anderen Größen gesetzt. Die Messungen solcher Metriken basieren dementsprechend auf der reinen Anzahl ausgewählter Entitäten und können auf verschiedenen System-Ebenen ermittelt werden. Dadurch kann die Größe eines gesamten Systems oder einzelner Komponenten, Klassen oder Methoden bezüglich der Vorkommnisse spezifischer Merkmale abgeschätzt werden.

Diese Metriken-Kategorie unterteilt sich wiederum in allgemeine Quantifizierungsmetriken, zeilenbasierte Quantifizierungsmetriken, aspekt-orientierte Quantifizierungsmetriken und merkmals-orientierte Quantifizierungsmetriken. Darüber hinaus existieren Quantifizierungsmetriken, welche nicht innerhalb dieser Metriken-Gruppen aufgelistet werden, da sie das Auftreten bestimmter Elemente erfassen, welche in anderen essenziell notwendig sind. Diese Metriken werden in ihren entsprechenden Metriken-Kategorien nochmal explizit hervorgehoben.

3.1.1.1 Allgemeine Quantifizierungsmetriken

Diese Unterkategorie enthält alle Quantifizierungsmetriken, welche die Eigenschaften der anderen Unterkategorien nicht erfüllen. Die darin enthaltenen Metriken wurden grundlegend für die objektorientierte Programmierung definiert. Sie können allerdings auch für Software-Projekt anderer Programmierparadigmen verwendet werden.

Eine Übersicht aller allgemeinen Quantifizierungsmetriken kann Tabelle 4 entnommen werden.

3.1.1.2 Zeilenbasierte Quantifizierungsmetriken

Die Unterkategorie der zeilenbasierten Quantifizierungsmetriken enthalten alle Metriken, welche die Zeilen einer vollständigen Anwendung oder einzelner Komponenten, Klassen oder Methoden zählen. Dabei kann das Zählen von Zeilen an bestimmte Bedingungen verknüpft sein. Die bekannteste zeilenbasierte Quantifizierungsmetrik wird als LOC bezeichnet und zählt bedingungslos alle Zeilen des Quellcodes einer Anwendung. Die Quantifizierungsmetrik $BLOC$ zählt hingegen nur alle Zeilen, welche keine Zeichen enthalten und dementsprechend leer sind. Zeilenbasierte Quantifizierungsmetriken, welche die Zeilen aspekt- oder merkmals-spezifischer Entitäten zählen, sind in dieser Unterkategorie jedoch nicht enthalten.

Eine Übersicht aller zeilenbasierten Quantifizierungsmetriken kann Tabelle 5 entnommen werden.

3.1.1.3 Aspekt-orientierte Quantifizierungsmetriken

Aspekt-orientierte Quantifizierungsmetriken zählen alle Entitäten, welche exklusiv in Software-Projekten mit aspekt-orientierter Programmierung vorkommen. Dazu zählen beispielsweise Aspekte, Advices und Joinpoints. Darüber hinaus gehören Quantifizierungsmetriken zu dieser Unterkategorie, sobald eine solche Metrik die Größe einer aspekt-spezifischen Entität abschätzen soll. Ein Beispiel dafür ist die Metrik AS, welche die Größe eines Aspekts durch die Zählung aller darin befindlichen Attribute, Methoden, Pointcuts, Advices und Deklarationen ermittelt.

Eine Übersicht aller aspekt-orientierten Quantifizierungsmetriken kann Tabelle 6 entnommen werden.

3.1.1.4 Merkmals-orientierte Quantifizierungsmetriken

In dieser Unterkategorie sind alle Quantifizierungsmetriken enthalten, welche auf der Zählung von Entitäten merkmals-orientierter Programmierung basieren. Dazu zählen beispielsweise Merkmale, Konstanten, Annotationen oder auch Verfeinerungen von Methoden und Konstanten. Analog zu den aspekt-orientierten Quantifizierungsmetriken zählen zu den merkmals-orientierten Quatnifizierungsmetriken ebenfalls solche Metriken, welche die Größe merkmals-spezifischer Entitäten abschätzen sollen.

Eine Übersicht aller merkmals-orientierten Quantifizierungsmetriken kann Tabelle 7 entnommen werden.

3.1.2 Komplexitätsmetriken

Diese Metriken-Kategorie umfasst alle Software-Metriken, welche die Komplexität des Quellcodes einer Software auf verschiedenen System-Ebenen unter der Betrachtung verschiedener Aspekte analysieren. Der Begriff der Software-Komplexität wird aufgrund einer fehlenden theoretischen

Grundlage und mangelnder Verallgemeinerung kritisch betrachtet [3]. Dementsprechend existieren verschiedene Ansätze zur Messung der Software-Komplexität, für die unterschiedliche Metriken definiert wurden. Somit teilen sich die Metriken dieser Gruppe basierend auf ihrer zugrunde liegenden Berechnungsgrundlage in verschiedene Unterkategorien auf. Diese Unterkategorien werden im folgenden als allgemeine Komplexitätsmetriken, zyklomatische Komplexitätsmetriken, Halstead-Komplexitätsmetriken, aspekt-orientierte Komplexitätsmetirken und merkmals-orientierte Komplexitätsmetriken bezeichnet.

3.1.2.1 Allgemeine Komplexitätsmetriken

Unter den allgemeinen Komplexitätsmetriken werden alle Metriken zusammengefasst, welche die Software-Komplexität aufgrund unterschiedlicher, hinreichender Kriterien erfassen. Darunter zählt beispielsweise die Metrik *HK*, welche als Henry-Kafura-Größe bekannt ist und die Komplexität des Kontrollflusses anhand quantitativer Größen berechnet [4].

Eine Übersicht aller allgemeinen Komplexitätsmetriken kann Tabelle 8 entnommen werden.

3.1.2.2 Zyklomatische Komplexitätsmetriken

Die zyklomatischen Komplexitätsmetriken basieren auf der Verschachtelung des Kontrollflusses einer Software-Anwendung. Dieser Ansatz basiert auf die Annahme, dass die Komplexität einer Software-Komponente mit zunehmender Verschachtelungstiefe steigt. Somit weist eine Komponente mit vielen Verzweigungen eine hohe Komplexität auf. Eine rein sequenzieller Kontrollfluss verfügt hingegen über eine sehr niedrige Komplexität. Dieser Komplexitätsansatz wurde ursprünglich im Jahr 1976 entworfen, auf dessen Grundlage die Metrik der zyklomatischen Komplexität *VG* durch Thomas J. McCabe veröffentlicht wurde [5].

Eine Übersicht aller zyklomatischen Komplexitätsmetriken kann Tabelle 9 entnommen werden.

3.1.2.3 Halstead-Komplexitätsmetriken

Neben den zyklomatischen Komplexitätsmetriken definierte Maurice Halstead im Jahr 1977 einen weiteren Ansatz zur Messung der Software-Komplexität. Dieser Ansatz basiert auf der Häufigkeit der Vorkommnisse von Operanden und Operatoren innerhalb einer Software-Einheit [6]. Als Grundlage der Halstead-Komplexität wurden die Quantifizierungsmetriken *NDO1*, *NDO2*, *TNO1* und *TNO2* definiert, weswegen diese Metriken in diese Kategorie eingeteilt wurden.

Eine Übersicht aller Halstead-Komplexitätsmetriken kann Tabelle 10 entnommen werden.

3.1.2.4 Kognitive Komplexitätsmetriken

Ähnlich zu den zyklomatischen Komplexitätsmetriken analysieren kognitive Komplexitätsmetriken den Kontrollfluss von Software-Anwendungen. Im Gegensatz zu der Verschachtelungstiefe des Kontrollflusses werden im Rahmen dieses Ansatzes die Basiskontrollstrukturen der einzelnen Software-Einheiten betrachtet. Dabei wird grundsätzlich zwischen Sequenzen, Verzweigungen, Iterationen, eingebetteten Komponenten und asynchronen Aufrufen als Arten von Basiskontrollstrukturen unterschieden, denen unterschiedliche Gewichtungen zugewiesen werden. Anhand der Vorkommnisse einzelner Basiskontrollstrukturen in Kombination mit deren Gewichtungen ergibt sich letztendlich ein kognitives Komplexitätsmaß. Mithilfe dieser Metriken sollen vor allen Software-Entwicklern einen Überblick über den benötigten kognitiven Aufwand für das Verständnis einzelner Software-Komponenten gegeben werden.

Eine Übersicht aller kognitiven Komplexitätsmetriken kann Tabelle 11 entnommen werden.

3.1.2.5 Aspekt-orientierte Komplexitätsmetriken

Die Unterkategorie der aspekt-orientierten Komplexitätsmetriken enthält Metriken, welche die Komplexität aspekt-spezifischer Entitäten ermitteln. So wird beispielsweise durch die Metrik *WOM* die Komplexität eines aspekt-orientierten Moduls anhand darin enthaltener Advices und Aspekten ermittelt.

Eine Übersicht aller aspekt-orientierten Komplexitätsmetriken kann Tabelle 12 entnommen werden.

3.1.2.6 Merkmals-orientierte Komplexitätsmetriken

Die Unterkategorie der merkmals-orientierten Komplexitätsmetriken enthält analog zur Unterkategorie der aspekt-orientierten Komplexitätsmetriken alle Metriken, welche die Komplexität merkmals-spezifischer Entitäten ermitteln. Durch die Metrik *AND* wird beispielsweise die durchschnittliche Verscahchtelungstiefe von Annotationen ermittelt.

Eine Übersicht aller merkmals-orientierten Komplexitätsmetriken kann Tabelle 13 entnommen werden.

3.1.3 Kohäsionsmetriken

Software-Projekte mit hoch kohäsiven Modulen sind in der Regel verständlicher, anpassbarer und wartbarer konstruiert. Solche Module implementieren in der Regel nur einzelne Basis-Funktion und können schwierig in mehrere Module aufgeteilt werden [7]. Zur Repräsentation dieser Eigenschaften stellen Kohäsionsmetriken ein Maß des Verwandtschaftsgrads von Elementen eines Moduls dar [8]. Diese intuitive Definition von Kohäsionsmetriken lässt jedoch einen großen Interpretationsspielraum zu, sodass viele verschiedene Metriken dieser Kategorie unterschiedliche Aspekte zur Relfektion möglicher Kohäsion-Kriterien erfassen.

Gewöhnlicherweise finden die zur Berechnung von Kohäsionsmetriken benötigten Messungen auf Klassen-Ebene statt. Dort stellt die Kohäsion den Verwandtschaftsgrad von Klassen-Attributen und -Methoden dar. Dabei basiert die Verwandtschaft dieser Elemente auf verschiedenen Arten von Interaktionen [7]. Zum einen werden Interaktionen zwischen Methoden berürcksichtigt, welche durch den Zugriff gleicher Attribute oder den gezielten Methodenaufruf entstehen. Zum anderen können Interaktionen zwischen Methoden und Attributen erfasst werden, die durch den Zugriff eines Attributs durch eine Methode ausgelöst werden. Weiterhin existieren Interkationen zwischen Attributen, sofern verschiedene Attribute durch die gleiche Methode aufgerufen werden.

Einige Kohäsionsmetriken erfassn sogar transitive Beziehungen dieser Interaktionen. Die meisten Kohäsionsmetriken erfassen jedoch nur strukturelle Merkmale auf Grundlage der Referenzierung von Klassen-Variablen zwischen den Methoden einer Klasse. Alle Kohäsionsmetriken erfassen jedoch ähnliche Eigenschaften. Dadurch eignet sich vor allem eine Unterteilung anhand zugrunde liegender Programmierparadigmen. Aufgrund der Vielzahl objekt-orientierte Kohäsionsmetriken wird diese Metriken-Teilmenge zusätzlich anhand zwei verschiedener Design-Ebenen der Entwurfsphase. Diese Metriken-Kategorie unterteilt sich somit zum einen in Kohäsionsmetriken des High-Level-Designs und des Low-Level-Designs und zum anderen in aspekt-orientierte Kohäsionsmetriken und merkmals-orientierte Kohäsionsmetriken.

3.1.3.1 Kohäsionsmetriken des High-Level-Designs

Kohäsionsmetriken des High-Level-Designs basieren auf Informationen bezüglich der Schnittstellen von Klassen und Methoden [7]. Die Abschätzung der Kohäsion findet durch diese Metriken demnach auf Grundlage nicht präzise definierter Informationen statt [9]. Interaktionen solcher Metriken basieren dadurch zum Großteil auf dem Vergleich von Datentypen von Attributen und Parametern.

Eine Übersicht aller Kohäsionsmetriken des High-Level-Designs kann Tabelle 14 entnommen werden.

3.1.3.2 Kohäsionsmetriken des Low-Level-Designs

Im Gegensatz zu den Kohäsionsmetriken des High-Level-Designs verwenden die Kohäsionsmetriken des Low-Level-Desings fein-granulare und präzise Informationen. Dazu werden entweder die in Klassen-Methoden verwendeten Algorithmen oder sehr präzise Informationen über Nachbedingungen dieser Methoden analysiert [7]. Aufgrund dieses höheren Informationsumfangs enthält diese Gruppe der Kohäsionsmetriken außerdem eine höhere Anzahl an Metriken.

Eine Übersicht aller Kohäsionsmetriken des Low-Level-Designs kann Tabelle 15 entnommen werden.

3.1.3.3 Aspekt-orientierte Kohäsionsmetriken

Die Unterkategorie der aspekt-orientierten Kohäsionsmetriken enthält Metriken, welche die Kohäsion aspekt-spezifischer Entitäten ermitteln. So wird beispielsweise durch die Metrik $ACOH$ die Kohäsion eines Aspekts auf Grundlage des Verwandtschaftsgrads verwendeter Klassen ermittelt.

Eine Übersicht aller aspekt-orientierten Kohäsionsmetriken kann Tabelle 16 entnommen werden.

3.1.3.4 Merkmals-orientierte Kohäsionsmetriken

Die Unterkategorie der merkmals-orientierten Kohäsionsmetriken enthält Metriken, welche die Kohäsion merkmals-spezifischer Entitäten ermitteln. Die in dieser Unterkategorie enthaltenen Metriken repräsentieren die Kohäsion von Merkmalen auf der Grundlage bestehender und potenzieller Abhängigkeiten.

Eine Übersicht aller merkmals-orientierten Kohäsionsmetriken kann Tabelle 17 entnommen werden.

3.1.4 Kopplungsmetriken

Kopplungsmetriken beschreiben die Unabhängigkeit zwischen Methoden und Objekt-Klassen [10]. Diese Unabhängigkeit wird anhand der Stärke von Assoziationen als Verbindungen zwischen verschiedenen Modulen eines Systems berechnet. Somit dienen diese Metriken für ein verbessertes Verständnis der Beziehungen und Interaktionen zwischen den einzelnen Komponenten eines Systems.

Ein System ohne Kopplung ist aufgrund fehlender Interaktionen zwischen seinen Modulen nutzlos. Daher wird stets ein gewisses Maß an Kopplung in jeder Anwendung benötigt. Unnötige Kopplung führt hingegen zur Instabilität eines System und der Reduzierung seiner Performanz. Software-Anwendungen mit hoher Kopplung sind komplexer aufgrund der erhöhten Verknüpfung verschiedener Komponenten Dadurch wird ein höheres Verständnis benötigt, um diese Software

zu implementieren, zu testen und zu warten. Daher sollte eine Software-Anwendung stets hohe Kohäsion und niedrige Kopplung vorweisen.

Die Berechnung der Unabhängigkeit zwischen verschiedenen Modulen basiert in der Regel auf der Erfassung von Methodenaufrufen und den Referenzierungen von Attributen [11]. Diese Interaktionen finden für gewöhnlich zwischen Klassen und Attributen, Klassen und Methoden oder zwischen verschiedenen Methoden statt. Weitere Interaktionen existieren ebenfalls als Abhängigkeiten zwischen einzelnen Elementen, sodass die Kopplung auch als Kombination verschiedener Arten von Interaktionen erfassbar sind. Darüber hinaus kann auch zwischen Import-Kopplung und Export-Kopplung unterschieden werden.

Eine Strukturierung dieser Metriken-Kategorie eignet sich daher vor allem anhand der verschiedenen Arten an Interaktionsarten. Daher werden alle Kopplungsmetriken in allgemeine Kopplungsmetriken, Kopplungsmetriken auf Grundlage von Datentypen und Feldzugriffen sowie auf Grundlage von Methodenaufrufen unterschieden. Weiterhin besteht eine Unterkategorie der apsektorientierten Kopplungsmetriken zur Zusammenfassung aller Kopplungsmetriken dieses Programmierparadigmas.

3.1.4.1 Allgemeine Kopplungsmetriken

In dieser Unterkategorie der Kopplungsmetriken werden alle Kopplungsmetriken zusammengefasst, welche die Unabhängigkeit zwischen Modulen auf der Grundlage allgemeiner oder spezifischer Interaktionen berechnen. Kopplungsmetriken basierend auf allen Arten von Interaktionen betrachten in der Regel alle möglichen Abhängigkeitsbeziehungen, welche zwischen verschiedenen Modulen möglich sind. Die Metrik *CBO* berechnet beispielsweise die Kopplung zwischen Klassen basierend auf Methodenaufrufen, Feldzugriffen, Vererbung, Argumenten, Rückgabewerten und Ausnahmebehandlungen. Darüber hinaus befinden sich innerhalb dieser Klasse Kopplungsmetriken, welche die Kopplung basierend auf speziellen Interaktionen berechenen. Die Metrik *MPC* berechnet beispielsweise die Kopplung einer Klasse auf Grundlage aller Nachrichten, welche zu anderen Klassen gesendet werden.

Eine Übersicht aller allgemeinen Kopplungsmetriken kann Tabelle 18 entnommen werden.

3.1.4.2 Kopplungsmetriken basierend auf Datentypen und Feldzugriffen

Diese Unterkategorie enthält alle Kopplungsmetriken, die entweder auf Felder anderer Klassen zugreifen oder andere Klassen als Datentypen verwenden. Der Feldzugriff anderer Klassen ähnelt dabei den Interaktionen zwischen Methoden und Attributen der Kohäsionsmetriken. Dazu werden jedoch nicht die Interaktionen zwischen den Methoden und Attributen der gleichen Klasse berücksichtigt. Stattdessen wird stets die Verwendung von Attributen anderer Klassen betrachtet.

Diese Interaktionen können wiederum als Export- oder Import-Kopplungen erfasst werden. Die Metrik *OCAIC* erfasst beispielsweise die Anzahl aller Klassen, welche andere Klassen als Datentyp eines Attributs enthalten. Die Metrik *OCAEC* erfasst wiederum die Anzahl aller Klassen, welche in einer anderen Klasse als Datentyp verwendet wurden.

Eine Übersicht aller Kopplungsmetriken basierend auf Datentypen und Feldzugriffen kann Tabelle 19 entnommen werden.

3.1.4.3 Kopplungsmetriken basierend auf Methodenaufrufen

Diese Kopplungsmetriken basieren auf dem Aufruf von Methoden anderer Klassen. Diese Art der Interaktion ist ähnlich zu den Methoden-Methoden-Interaktionen der Kohäsionsmetriken. Es werden jedoch nur die Methodenaufrufe betrachtet, welche zwischen verschiedenen Klassen ausgelöst werden.

Identisch zu den auf Datentypen und Feldzugriffen basierenden Kopplungsmetriken kann in dieser Unterkategorie ebenfalls zwischen Import- und Export-Kopplung unterschieden werden. So berechnet die Metrik *FANIN* beispielsweise die Anzahl an Klassen, welche die Methoden einer bestimmten Klasse aufrufen. Die Metrik *FANOUT* erfasst hingegen die Anzahl aller Klassen, in denen die Methoden einer bestimmten Klasse aufgerufen werden.

Eine Übersicht aller Kopplungsmetriken basierend auf Methodenaufrufen kann Tabelle 20 entnommen werden.

3.1.4.4 Aspekt-orientierte Kopplungsmetriken

Die Unterkategorie der aspekt-orientierten Kopplungssmetriken enthält Metriken, welche die Kopplung aspekt-spezifischer Entitäten ermitteln. Aufgrund der Einführung neuer Entitäten durch dieses Programmierparadigma entsteht eine Vielfalt neuer Arten von Interaktionen und Abhängigkeiten zwischen aspekt-orientierten Modulen. Durch die Metrik *BAC* wird zum Beispiel die Kopplung eines Aspekts mithilfe der Anzahl der durch Advices versteckten Joinpoints und der durch verschiedene Deklarationen möglichen Veränderungen der Modul-Hierarchie berechnet.

Eine Übersicht aller aspekt-orientierten Kopplungsmetriken kann Tabelle 21 entnommen werden.

3.1.5 Vererbungsmetriken

Vererbung ist ein mächtiges Konzept der objekt-orientierten Programmierung, welches bereits im Jahr 1967 eingeführt wurde [12]. Mittels Vererbung können Beziehungen zwischen Klassen definiert werden, um die Wiederverwendbarkeit von Software-Komponenten zu erhöhen. Dazu werden Unterklassen von Superklassen abgeleitet, sodass die in den Superklassen enthaltenen Methoden und Attribute für Unterklassen verfügbar sind. Innerhalb der Unterklassen können wiederum zusätzliche Attribute und Methoden implementiert werden, welche gegebenenfalls Funktionalitäten abgeleiteter Superklassen verwenden.

Durch mehrfache Verschachtelung von Vererbungsstrukturen kann jedoch die Komplexität einer Software-Anwendung signifikant erhöht werden. Daher muss ein Kompromiss zwischen der Komplexität und Wiederverwendbarkeit einer Software getroffen werden, um eine hohe Qualität des Produkts zu gewährleisten. In diesem Kontext spielen Vererbeungsmetriken eine zentrale Rolle, da sie den Umfang an Wiederverwendbarkeit von Methoden und Klassen basierend auf Vererbung messen [13].

Dazu existieren viele Vererbungsmetriken, welche einzelne Eigenschaften von Vererbungsbeziehungen auf Klassen-Ebene repräsentieren. Weitere Vererbungsmetriken betrachten wiederum die Wiederverwendbarkeit geteilter Attribute und Methoden oder basieren auf der Auswertung der gesamten Vererbungshierarchie. Dementsprechend wurde diese Metriken-Kategorie anhand der Granularität zugrunde liegender Informationen unterteilt. So existieren die Unterkategorien der Vererbungsmetriken auf Klassen-Ebene, die Vererbungsmetriken der Methoden- und Attribut-Ebene

sowie Metriken der gesamten Vererbungshierarchie. Darüber hinaus wurde eine Unterkategorie der aspekt-orientierten Vererbungsmetriken definiert.

3.1.5.1 Vererbungsmetriken der Klassen-Ebene

Die Vererbungsmetriken der Klassen-Ebene beschreiben den Vererbungsgrad einzelner Klassen aufgrund ihrer direkten und indirekten Vererbungsbeziehungen. Die meisten dieser Metriken gelten ebenfalls als Quantifizierungsmetriken, da sie lediglich die Anzahl vererbter oder geerbter Entitäten ermitteln. So berechnet beispielsweise die Metrik *FANUP* die Anzahl aller Superklassen einer gegebenen Klasse und die Metrik *FANDOWN* die Anzahl aller Unterklassen einer gegebenen Klasse.

Eine Übersicht aller Vererbungsmetriken der Klassen-Ebene kann Tabelle 22 entnommen werden.

3.1.5.2 Vererbungsmetriken der Attribut- und Methoden-Ebene

Die Vererbungsmetriken der Attribut- und Methoden-Ebene beschreiben den Grad der Vererbung von Klassen basierend ihren Attributen und Methoden. Ebenso wie die Vererbungsmetriken der Klassen-Ebene gelten die meisten Vererbungsmetriken der Attribut- und Klassen-Ebene als Quantifizierungsmetriken. Die Metrik *NMO* erfasst beispielsweise die Anzahl aller geerbten Methoden einer Klasse, die überschrieben worden. Die Metrik *MFA* beschreibt hingegen ein Verhältnis zwischen der Anzahl aller geerbten Methoden einer Klasse und allen verfügbaren Methoden dieser Klasse.

Eine Übersicht aller Vererbungsmetriken der Attribut- und Methoden-Ebene kann Tabelle 23 entnommen werden.

3.1.5.3 Metriken der Vererbungshierarchie

Die Vererbungsmetriken dieser Metriken-Kategorie basieren auf der vollständigen oder teilweisen Auswertung der Vererbungshierarchie. In der Regel werden dazu alle Vererbungsbeziehungen durch einen Vererbungsbaum repräsentiert. Mithilfe dieses Baums können zusätzliche Informationen zur Beschreibung des Vererbungsgrads einzelner Klassen oder des gesamten Software-Systems erfasst werden, welche durch die Auswertung einzelner Entitäten nicht möglich ist. Die Metrik *DIT* ermittelt beispielsweise die maximale Tiefe eines solchen Vererbungsbaums. Die Metrik *DBRM* berechnet hingegen das Verhältnis zwischen der Anzahl aller abgeleiteten Klassen und allen Basisklassen des Vererbungsbaums.

Eine Übersicht aller Metriken der Vererbungshierarchie kann Tabelle 24 entnommen werden.

3.1.5.4 Aspekt-orientierte Vererbungsmetriken

Die Unterkategorie der aspekt-orientierten Vererbungssmetriken enthält Metriken, welche die Vererbung aspekt-spezifischer Entitäten ermitteln. Diese Metriken berechnen den Vererbungsgrad von Aspekten, Advices, Pointcuts und Attributen. Die Metrik *APIF* berechnet beispielsweise das Verhältnis zwischen der Anzahl aller konkreten Aspekte und allen verfügbaren Aspekten. Die Metrik *ADIF* berechnet hingegen das Verhältnis zwischen der Anzahl neudefinierter Advices und allen verfügbaren Advices.

Eine Übersicht aller aspekt-orientierten Vererbungsmetriken kann Tabelle 25 entnommen werden.

3.1.6 Sicherheitsmetriken

Mittels Sicherheitsmetriken wird die Angriffsoberfläche von Software-Anwendungen beschrieben. Diese Angriffsoberfläche kann durch die Summe aller Eingangs- und Ausgangspunkte einzelner Software-Komponenten repräsentiert werden [14]. Eine obere Schranke dieser Quantifizierung stellt beispielsweise die Anzahl aller Attribute und Methoden dar. Durch die Kapselung einzelner Entitäten kann diese Fläche jedoch reduziert werden, sodass nur noch öffentlich verfügbare Attribute und Methoden ein potenzielles Sicherheitsrisiko bilden.

Daher werden alle Sicherheitsmetriken grundlegend in Zugriffsmetriken und Kapselungsmetriken unterteilt. Darüber hinaus existiert eine weiter Unterkategorie der Sicherheitsmetriken, welche Informationen bezüglich Ausnahmebehandlungen erfassen. Diese Metriken können zusätzliche Sicherheitsrisiken erzeugen, sodass sie ebenfalls als Bestandteil der Sicherheitsmetriken angesehen werden können.

3.1.6.1 Zugriffsmetriken

Zugriffsmetriken dienen zur Quantifizierung und Beschreibung der Zugriffsmöglichkeiten auf verschiedene Attribute und Methoden. Die meisten Zugriffsmetriken gelten daher auch als Quantifizierungsmetriken repräsentieren lediglich die Anzahl an Attributen oder Methoden, welche mit einem bestimmten Zugriffsmodifikator markiert wurden. Die Metrik *NOPRA* ermittelt beispielsweise die Anzahl aller als privat deklarierten Attribute einer Klasse.

Eine Übersicht aller Zugriffsmetriken kann Tabelle 26 entnommen werden.

3.1.6.2 Kapselungsmetriken

Kapselungsmetriken fassen alle Software-Metriken zusammen, welche die Funktionalitäten oder Eigenschaften einzelner Entitäten verschleiern. Solche Elemente sind nur innerhalb der jeweiligen Klasse verfügbar. Entsprechende Attribute können beispielsweise nur durch die Definition so genannter Getter- und Setter-Methoden aufgerufen und verändert werden, um Sicherheitsrisiken zu minimieren.

Eine Übersicht aller Kapselungsmetriken kann 27 entnommen werden.

3.1.6.3 Metriken der Ausnahmebehandlung

Metriken der Ausnahmebehandlung umfassen alle Metriken, welche Informationen bezüglich der Behandlung von Ausnahmen bereitstellen. Die meisten dieser Metriken erfassen die Anzahl bestimmter Komponenten der Ausnahmebehandlung und gelten daher ebenfalls als Quantifizierungsmetriken. Die Metrik *NOCB* repräsentiert beispielsweise die Anzahl aller Catch-Blöcke einer Software-Komponente oder des gesamten Systems.

Eine Übersicht aller Metriken der Ausnahmebehandlung kann 28 entnommen werden.

3.1.7 Netzwerkmetriken

Netzwerkmetriken dienen zur Beschreibung von Abhängigkeitsstrukturen zwischen den einzelnen Code-Elementen einer Software-Anwendung [15]. Diese Metriken werden dazu in der Regel von einem Abhängigkeitsgraphen abgeleitet. Innerhalb eines solchen Graphen werden alle Entitäten einer Software-Anwendung als Knoten dargestellt und durch gerichtete Abhängigkeitsbeziehungen

als Kanten miteinander verknüpft. Dabei unterscheidet man im Wesentlichen zwischen datenbezogenen und aufrufs-bezogenen Abhängigkeiten. Während Datenabhängigkeiten die Deklaration und die Verwendung von Werten betrachten, beziehen sich Aufrufsabhängigkeiten auf der Deklaration und dem Aufruf von Elementen. Somit können die Kanten eines Abhängigkeitsgraphen beispielswiese eine Beziehung zwischen einer aufrufenden und einer aufgerufenen Entität repräsentieren.

Für eine Software-Anwendung können so genannte Ego-Netzwerke und ein globales Netzwerk erstellt werden. Diese Netzwerke definieren unterschiedliche Mengen von Netzwerkmetriken, um die darin enthaltenen Informationen zu erfassen. Darüber hinaus existieren weitere Netzwerkmetriken zur Beschreibung der Balancierung eines solchen Graphen. Deshalb unterteilen sich alle Netzwerkmetriken in Metriken des Ego-Netzwerks, globale Netzwerkmetriken, Metriken struktureller Löcher und Zentralitätsmetriken.

3.1.7.1 Metriken des Ego-Netzwerks

Ein Ego-Netzwerk beschreibt die direkten Abhängigkeiten zwischen einem ausgewählten Element und seiner unmittelbaren Nachbarschaft im Abhängigkeitsgraphen [16]. Mithilfe dieser Metriken sollen die direkte Auswirkungen zwischen miteinander verknüpften Elementen betrachtet werden, um die Relevanz der ausgewählten Elemente zu erfassen. Die Metrik *SIZE* berechnet beispielsweise die Anzahl aller Knoten eines Ego-Netzwerks. Die Metrik *TIES* ermittelt hingegen die Anzahl aller Kanten eines Ego-Netzwerks.

Eine Übersicht aller Metriken des Ego-Netzwerks kann Tabelle 29 entnommen werden.

3.1.7.2 Globale Netzwerkmetriken

Im Gegenatz zu den Metriken des Ego-Netzwerks betrachten globale Netzwerkmetriken alle im vollständigen Abhängigkeitsgraphen enthaltenen Abhängigkeiten [16]. Dementsprechend basieren globale Netzwerkmetriken auf der gesamten Abhängigkeitsstruktur eines Software-Systems. Sie dienen zur Messung der Reichweite von Auswirkungen der Abhängigkeiten einzelner Elemente und erfassen deren Relevanz innerhalb des gesmaten Systems. Die Metrik N ermittelt beispielsweise die Anzahl aller Knoten des globalen Netzwerks. Die Metrik L erfasst hingegen die Anzahl aller Kanten des globalen Netzwerks.

Eine Übersicht aller globalen Netzwerkmetriken kann Tabelle 30 entnommen werden.

3.1.7.3 Metriken struktureller Löcher

Die Metriken struktureller Löcher untersuchen den Einfluss von Entitäten im Verhältnis zu gut balancierten Netzwerken [16]. Ein gut balanciertes Netzwerk entsteht durch die Verknüpfung aller Entitäten durch Abhängigkeiten. Entitäten mit fehlenden Abhängigkeiten zu anderen Entitäten können über einen höheren oder niedrigeren Einfluss auf andere Entitäten verfügen. Daher werden durch die Metriken strukturelle Löcher das Fehlen entsprechender Abhängigkeitsbeziehungen erfasst.

Eine Übersicht aller Metriken struktureller Löcher kann Tabelle 31 entnommen werden.

3.1.7.4 Zentralitätsmetriken

Ähnlich zu den Metriken struktureller Löcher betrachten Zentralitätsmetriken den Einfluss von Entitäten im Verhältnis zu gut balancierten Netzwerken. Im Gegensatz zu den Metriken struktu-

reller Löcher wird das Netzwerk jedoch dahingehend überprüft, wie sehr es einem gut balancierten Netzwerk ähnelt.

Eine Übersicht aller Zentralitätsmetriken kann Tabelle 32 entnommen werden.

3.2 Verwendung von Software-Metriken

Nicht alle Metriken dieser Taxonomie wurden unter allen möglichen Aspekt gleichmäßig untersucht. Viele dieser Metriken wurden in verschiedenen Bereichen der Software-Entwicklung zur Lösung spezifischer Apsekte verwendet. Daher müssen zur Erstellung eines angemessenen Software-Qualitätsmodells geeignete Metriken aus der Taxonomie gewählt werden. Durch eine Beschränkung der Auswahl verwendeter Metriken profitiert außerdem die Performanz dieses Modells, während gleichzeitig Gefahren der Überanpassung reduziert werdne können.

Die am meisten untersuchten Metriken des objekt-orientierten Paradigma sind die von Chidamber und Kemerer erstellten Metrik sowie *LOC* und die zyklomatischen Komplexitätsmetriken von McCabe. Diese Metriken wurden bereits vor 1970 das erste Mal verwendet und gewannen durch die steigende Popularität der objekt-orientierten Programmierung an Interesse. Seitdem wurden diese Metriken zur Messung und Erhöhung der Qualität in einer Vielfalt von Anwendungen eingesetzt. Die Metriken von Chidamber und Kemerer gelten dabei als Pioniere der Forschung und repräsentieren die theoretische Basis zur Messung objekt-orientierten Quellcodes. Mithilfe dieser Metriken können die wichtigsten Attribute, wie Komplexität, Kohäsion und Kopplung abgedeckt werden. Aufgrund gewissenhafter Untersuchungen stellen diese Metriken somit die Basis weiterer entworfener Metriken, welche beispielsweise für die aspekt- und merkmals-orientierte Programmierung adaptiert worden.

Darüber hinaus wurden im Rahmen der systematischen Mapping-Studie die folgenden zehn am meisten verwendeten objekt-orientierten Software-Metriken identifiziert [1]:

- Gewichtete Methoden pro Klasse (Weighted Methods per Class)
- Kopplung zwischen Objekten (Coupling Between Objects)
- Mangel an Kohäsion in Methoden 1 (Lack of Cohesion in Methods 1)
- Tiefe des Vererbungsbaums (Depth of Inheritance Tree)
- Anzahl Code-Zeilen (Lines of Code)
- Anzahl an Kindern (Number of Children)
- Antworten für eine Klasse (Response for a class)
- Anzahl an Klassen-Methoden (Number of Class Methods)
- McCabe's zyklomatische Komplexität (McCabe Cyclomatic Complexity)
- Anzahl an Attributen (Number of Attributes)

Weitere häufig verwendete Metriken lauten *FANIN*, *FANOUT*, *NOPM*, *LCOMM*, *CA*, *CE* und *SLOC*. Die Verwendung der 10 am meisten untersuchten Metriken ist jedoch signifikant höher im Vergleich zu den restlichen Metriken. So wurden alle weiteren objekt-orientierten Metriken sehr selten oder sogar nur einmalig verwendet.

Paradigma	Metriken
Aspekt-orientiert	• Zerstreuung eines Concerns über Komponenten (Concern Diffusion over Components) • Zerstreuung eines Concerns über Operationen (Concern Diffusion Over Operations) • Mangel an Kohäsion in Operationen (Lack of Cohesion in Operations) • Kopplung zwischen Komponenten (Coupling Between Components)
Merkmals-orientiert	• Anzahl an Merkmalen (Number of Features) • Anzahl verfeinerter Konstanten (Number of Refined Constants)
Prozedural	• McCabe's zyklomatische Komplexität (McCabe Cyclomatic Complexity) • Anzahl Code-Zeilen (Lines of Code)

Tabelle 2: Am meisten verwendete Software-Metriken verschiedener Programmierparadigmen [1]

Für alle weiteren Programmierparadigmen wurden laut der systematischen Mapping-Studie die in Tabelle 2 aufgelisteten Software-Metriken am meisten verwendet [1].

Die Aufzählungen repräsentieren die Häufigkeit der Verwendung von Software-Metriken im Allgemeinen. Da die geeignete Metriken zur Erstellung eines Software-Qualitätsmodells ausgewählt werden sollen, werden in Tabelle 3 alle Metriken aufgezählt, die in den Bereichen der Software-Qualität und Fehlererkennung verwendet wurden [1]. Dabei konnten keine merkmals-orientierten Software-Metriken identifiziert werden, welche in diesen Bereichen eingesetzt wurden.

Paradigma	**Metriken**
Objekt-orientiert	A, ACAIC, ACMIC, AHF, AID, AIF, AMC, AMMIC, ANA, ATFD, BLOC, CA, CAAEC, CAM, CBI, CBM, CBMC, CBO, CCOH, CE, CIF, CINT, CIS, CHC, CLD, CM, CO, COF, COH, COM, DAC, DAM, DCAEC, DCC, DCD, DCI, DCMEC, DIT, DMMEC, DN, DSC, ELOC, EVG, HK, HSE, ICBMC, ICH, ICP, IVG, LCC, LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, LCOMM, LOC, LOCEH, LSCC, LVAR, MFA, MHF, MIF, MLOC, MMAC, MOA, MPC, NCLASS, NDO1, NDO2, NEST, NHD, NIM, NINT, NMI, NMO, NMS, NOA, NOAC, NOAI, NOAM, NOAV, NOCB, NOD, NOFB, NOH, NOM, NOPA, NOPM, NOPRA, NOPRM, NOTB, NP, NPAR, NPATH, NSC, NSF, NSM, OCAEC, OCAIC, OCMEC, OCMIC, OLN, OMMEC, OMMIC, PCCC, PF, RFC, RMI, RR, SCOM, SDMC, SEIMI, SIX, SLOC, SR, SUBC, SUPC, SVG, TCC, TLCOM, TNO1, TNO2, VG, VOL, WCCF, WMC
Aspekt-orientiert	BAC, CAE, CDA, CFA, CMC, DIT, DOS, LCOO, NA, NOCOA, RFM, WOM
Prozedural	BLOC, CA, CE, HSD, HSE, LCOMM, LOC, NDO1, NDO2, PL, PV, TNO1, TNO2, VG, VOL

Tabelle 3: Am häufigsten zur Fehlererkennung und Bewertung der Software-Qualität verwendeten Metriken [1]

4 Fazit

Durch die Auswertung der erstellten Metriken-Taxonomie sollen die aufgestellten Forschungsfragen beantwortet werden. Daher wird in Kapitel 4.1 die Taxonomie inklusive ihres Entstehungsprozesses rückblickend zusammengefasst. Die daraus resultierenden Ergebnisse werden zur Beantwortung der Forschungsfragen in Kapitel 4.2 behandelt. Anschließend folgt in 4.3 basierend auf diesen Erkenntnissen ein kurzer Überblick weiterer Forschungsmöglichkeiten.

4.1 Zusammenfassung

Software-Produktmetriken dienen zur Abbildung der Charasteristiken von Entitäten als Eigenschaften eines Software-Produkts. Dazu werden Informationen aus dem Quellcode einer Software extrahiert und gegebenenfalls weiterverarbeitet. Die daraus resultierenden Werte werden meistens numerisch, ordinär oder kategorisch repräsentiert. Diese Metriken können wiederum zur Bewertung der Software-Qualität verwendet werden, was ein Bestandteil der Software-Messungsprozesses jeder Entwicklungsphase ist.

Zur Identifikation geeigneter Produktmetriken für den Einsatz in Software-Qualitätsmodellen wurde daher eine Literaturrecherche unter Anwendung des Schneeballsystems ausgeführt. Im Rahmen dieser Recherche wurden initial eine systematische Mapping-Studie von Quellcode-Metriken sowie eine Vergleichsstudie projektübergreifender Ansätze der Fehlervorhersage betrachtet. Die darin untersuchten und verwendeten Metriken wurden möglichst umfassend erfasst und durch zusätzliche Informationen aus referenzierten Quellen ergänzt. Durch die Kategorisierung aller ermittelten Software-Produktmetriken wurde außerdem eine Taxonomie erstellt. Mithilfe einzelner Metriken-Definitionen wurde diese Taxonomie auf Korrektheit und Redundanz überprüft, sodass gegebenenfalls Namensgebungen für Bezeichnungen und Abkürzungen angepasst werden mussten.

Die konkreten Inhalte dieser Taxonomie werden in Kapitel 4.2 ausgewertet.

4.2 Ergebnisse

Im Rahmen dieses Kapitels findet eine Auswertung der Taxonomie statt, sodass alle aufgestellten Forschungsfragen nacheinander beantwortet werden.

Die Forschungsfrage **RQ1** thematisiert die Identifikation verschiedener Produktmetriken inklusive ihrer Definitionen. Im Rahmen dieser Forschungsarbeit wurden 344 unterschiedliche Metriken ermittelt, welche für die verschiedenen Programmierparadigmen der prozeduralen, objekt-, apsekt- und merkmals-orientierten Programmierung konzipiert worden. Dabei wurden die ersten Metriken für die prozedurale Programmierung entwickelt, ehe die Beliebtheit der objekt-orientierten Programmierung seit 1990 erheblich anstieg. Dadurch wurden die Metriken der objekt-orientierten Programmierung ein zentrales Element der Forschung, sodass die meisten Produktmetriken für dieses Paradigma definiert worden. Die meisten aspekt- und merkmals-orientierten Metriken wurden auf Grundlage objekt-orientierter Metriken angepasst und adaptiert. Vor allem für die merkmals-orientierte Programmierung werden jedoch weitere Studien zur Ermittlung hilfreicher Metriken benötigt.

Eine Übersicht aller Software-Produktmetriken inklusive ihrer Definitionen kann Kapitel 7 entnommen werden.

Durch die Forschungsfrage **RQ2** sollen die verschiedenen Kategorien von Software-Metriken identifiziert werden, welche durch die Erstellung der Taxonomie eingeführt worden. Dazu wurden

grundlegend die folgenden sieben verschiedenen Metriken-Kategorien definiert:

- **Quantifizierungsmetriken**: Beschreibung der Größe einzelner Software-Komponenten basierend auf der Anzahl an Vorkommnissen von Entitäten mit bestimmten Eigenschaften
- **Komplexitätsmetriken**: Beschreibung der Komplexität einer Software-Anwendung basierend auf verschiedenen Ansätzen
- **Kohäsionsmetriken**: Beschreibung des Verwandtschaftsgrads von Elementen innerhalb eines Moduls basierend auf verschiedenen Arten von Interaktionen
- **Kopplungsmetriken**: Beschreibung der Unabhängigkeit zwischen Methoden und Objekt-Klassen auf Grundlage der Stärke von Assoziationen als Verbindungen zwischen den Modulen eines Systems
- **Vererbungsmetriken**: Beschreibung der Vererbungsbeziehungen innerhalb einzelner Software-Komponenten zur Abwägung zwischen Wiederverwendbarkeit und Komplexität
- **Sicherheitsmetriken**: Beschreibung der Angriffsoberfläche einer Software-Anwendung anhand von Eingangs- und Ausgangspunkten einzelner Software-Komponenten
- **Netzwerkmetriken**: Beschreibung von Abhängigkeitsstrukturen zwischen einzelnen Code-Elementen einer Software-Anwendung auf der Grundlage von Abhängigkeitsgraphen

Darüber hinaus verfügen diese Metriken-Kategorien über verschiedene Unterkategorien. Die Beschreibung dieser Unterkategorien kann den entsprechenden Unterkapiteln des Kapitels 3.1 entnommen werden.

Zur Beantwortung der Forschungsfrage **RQ3** wurden die ermittelten Software-Produktmetriken hinsichtlich ihrer Verwendung zur Bewertung von Software-Qualität und der Erkennung von Fehlern untersucht. Kapitel 3.2 enthält eine Übersicht aller Metriken, welche in diesen relevanten Bereichen eingesetzt worden. Im Vergleich zu der Menge der am meisten untersuchten Metriken repräsentiert diese Übersicht eher eine Erweiterung der Auflistung. Mit Ausnahme der aspekt-orientierten Metriken wurden alle häufig untersuchten obejkt-orientierten Metriken bereits im Rahmen der Qualitätsbewertung und Fehlererkennung verwendet. Für die aspekt-orientierte Programmierung wurden hingegen die Metriken *CDC* und *CDO* noch nicht in diesen Bereichen eingesetzt. Abgesehen von den Metriken der merkmals-orientierten Metriken wurde für jedes Paradigma jedoch eine viel höhere Vielfalt verschiedener Metriken verwendet. Der fehlende Einsatz merkmals-orientierter Metriken zur Bewertung von Software-Qualität und der Erkennung von Fehlern lässt sich dabei vermutlich auf Benötigung zusätzlicher Studien zurückführen.

4.3 Ausblick

Die im Rahmen dieser Forschungsarbeit erstellte Taxonomie kann als effizientes Werkzeug im Software-Messungsprozess verwendet werden. Mithilfe dieser Taxonomie kann aufgrund der Sammlung verschiedener Software-Produktmetriken ein grudlegender Überblick bereitgestellt werden. Dadurch kann die Auswahl verschiedener Metriken für den Einsatz in verschiedenen Arten von Modellen sowie in vielen weiteren Anwendungsfällen unterstützt werden. Darüber hinaus dient diese Übersicht als Einstiegspunkt zur Sammlung weiterer Informationen bezüglich der darin aufgelisteten Metriken.

Neben der erstellten Taxonomie bieten die zusätzlichen Informationen hinsichtlich der Verwendung von Software-Produktmetriken einen idealen Einstiegspunkt zur Erstellung eines Software-Qualitätsmodells. Alle in Kapitel 3.2 aufgelisteten Metriken wurden entweder tiefgründiger untersucht oder bereits für solche Anwendungsfälle eingesetzt. Daher sollten diese Metriken hinsichtlich ihrer Performanz in solchen Modelle überprüft werden, da die optimale Auswahl von Metriken auf der Grundlage empirischer Studien basiern sollte. Korrelationsanalysen von Metriken können beispielsweise ausgeführt werden, um den Zusammenhang verschiedener Metriken hinsichtlich der Qualitätsbewertung und Fehlererkennung einer Software zu messen. Die meisten Korrelationsanalysen verwenden jedoch verschiedene Metriken basierend auf untershciedlichen Datensätzen, sodass eine Vergleichbarkeit der Studien und letztendlich der verschiedenen Metriken kritisch ist. Dementsprechend würde sich eine umfangreichere Korrelationsanalyse auf einer vielfältigen Basis an Software-Produktmetriken und Datensätzen eignen, um konkretere Aussagen über den Einfluss einzelner Metriken treffen zu können.

Darüber hinaus können durch umfangreiche Tests die Performanz verschiedener Metriken-Kombinationen berechnen. Unter Verwendung der Taxonomie lassen sich dadurch beispielsweise die performantesten Metriken jeder einzelnen Kategorie bestimmen. Somit kann letztendlich durch eine optimale Auswahl entsprechender Metriken der maximal mögliche Einfluss auf die Bewertung der Software-Qualität und die Erkennung der Fehlern durch einzelne Metriken-Kategorien abschätzen.

5 Literaturverzeichnis

[1] Alberto S. Nuñez-Varela u. a. „Source code metrics: A systematic mapping study“. In: *Journal of Systems and Software* 128 (2017), S. 164–197. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2017.03.044. URL: https://www.sciencedirect.com/science/article/pii/S0164121217300663.

[2] Steffen Herbold, Alexander Trautsch und Jens Grabowski. „A Comparative Study to Benchmark Cross-Project Defect Prediction Approaches“. In: *IEEE Transactions on Software Engineering* 44.9 (2018), S. 811–833. DOI: 10.1109/TSE.2017.2724538.

[3] Sanjay Misra. „An Object Oriented Complexity Metric Based on Cognitive Weights“. In: *6th IEEE International Conference on Cognitive Informatics*. 2007, S. 134–139. DOI: 10.1109/COGINF.2007.4341883.

[4] Darko Durisic u. a. „Measuring the impact of changes to the complexity and coupling properties of automotive software systems“. In: *Journal of Systems and Software* 86.5 (2013), S. 1275–1293. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2012.12.021. URL: https://www.sciencedirect.com/science/article/pii/S0164121212003330.

[5] Yuming Zhou, Baowen Xu und Hareton Leung. „On the ability of complexity metrics to predict fault-prone classes in object-oriented systems“. In: *Journal of Systems and Software* 83.4 (2010), S. 660–674. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2009.11.704. URL: https://www.sciencedirect.com/science/article/pii/S0164121209002970.

[6] Peter G. Hamer und Gillian D. Frewin. „M.H. Halstead's Software Science - a Critical Examination“. In: *Proceedings of the 6th International Conference on Software Engineering*. ICSE '82. Tokyo, Japan: IEEE Computer Society Press, 1982, S. 197–206.

[7] Jehad Al Dallal und Lionel C. Briand. „An object-oriented high-level design-based class cohesion metric“. In: *Information and Software Technology* 52.12 (2010), S. 1346–1361. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2010.08.006. URL: https://www.sciencedirect.com/science/article/pii/S0950584910001552.

[8] A. Marcus und D. Poshyvanyk. „The conceptual cohesion of classes“. In: *21st IEEE International Conference on Software Maintenance (ICSM'05)*. 2005, S. 133–142. DOI: 10.1109/ICSM.2005.89.

[9] Jehad Al Dallal und Lionel Claude Briand. „A Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes“. In: *ACM Trans. Softw. Eng. Methodol.* 21 (2012), 8:1–8:34.

[10] K. M. Azharul Hasan und Mohammad Sabbir Hasan. „Principal component analysis of coupling measures for developing high quality object oriented software“. In: *International Conference on Computer and Communication Engineering (ICCCE'10)*. 2010, S. 1–6. DOI: 10.1109/ICCCE.2010.5556837.

[11] Denys Poshyvanyk u. a. „Using information retrieval based coupling measures for impact analysis“. In: *Empirical Software Engineering* 14 (2008), S. 5–32.

[12] N. B. Singh u. a. „Impact of shared attributes and methods in calculation of object-oriented inheritance metrics“. In: *2014 2nd International Conference on Emerging Technology Trends in Electronics, Communication and Networking*. 2014, S. 1–7. DOI: 10.1109/ET2ECN.2014.7044984.

[13] Usha Kumari und Sucheta Bhasin. „Application of Object-Oriented Metrics To C++ and Java: A Comparative Study“. In: *SIGSOFT Softw. Eng. Notes* 36.2 (Mai 2011), S. 1–10. ISSN: 0163-5948. DOI: 10.1145/1943371.1943386. URL: https://doi.org/10.1145/1943371.1943386.

[14] Mengyuan Zhang u. a. „Large-Scale Empirical Study of Important Features Indicative of Discovered Vulnerabilities to Assess Application Security“. In: *IEEE Transactions on Information Forensics and Security* 14.9 (2019), S. 2315–2330. DOI: 10.1109/TIFS.2019.2895963.

[15] Ayşe Tosun, Burak Turhan und Ayşe Bener. „Validation of Network Measures as Indicators of Defective Modules in Software Systems“. In: *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. PROMISE '09. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2009. ISBN: 9781605586342. DOI: 10.1145/1540438.1540446. URL: https://doi.org/10.1145/1540438.1540446.

[16] Kim Herzig u. a. „Predicting defects using change genealogies“. In: *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. 2013, S. 118–127. DOI: 10.1109/ISSRE.2013.6698911.

[17] Vibhash Yadav und Raghuraj Singh. „Predicting Design Quality of Object-Oriented Software using UML diagrams“. In: *2013 3rd IEEE International Advance Computing Conference (IACC)*. 2013, S. 1462–1467. DOI: 10.1109/IAdCC.2013.6514442.

[18] Emanuel Giger u. a. „Method-level bug prediction“. In: Sep. 2012, S. 171–180. ISBN: 978-1-4503-1056-7. DOI: 10.1145/2372251.2372285.

[19] Giuseppe Scanniello. „An Investigation of Object-Oriented and Code-Size Metrics as Dead Code Predictors“. In: *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. 2014, S. 392–397. DOI: 10.1109/SEAA.2014.67.

[20] Abdul Jabbar und Subramani Sarala. „Information flow metrics analysis in object oriented programming and metrics validation process by RAA algorithm“. In: *Advances in Engineering Software* 54 (2012), S. 30–37. ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2012.08.004. URL: https://www.sciencedirect.com/science/article/pii/S0965997812001147.

[21] Emad Shihab u. a. „Is lines of code a good measure of effort in effort-aware models?“ In: *Information and Software Technology* 55.11 (2013), S. 1981–1993. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2013.06.002. URL: https://www.sciencedirect.com/science/article/pii/S0950584913001316.

[22] Yonghee Shin und Laurie Williams. „An Empirical Model to Predict Security Vulnerabilities Using Code Complexity Metrics“. In: *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '08. Kaiserslautern, Germany: Association for Computing Machinery, 2008, S. 315–317. ISBN: 9781595939715. DOI: 10.1145/1414004.1414065. URL: https://doi.org/10.1145/1414004.1414065.

[23] Miltiadis Siavvas, Dionysios Kehagias und Dimitrios Tzovaras. „A Preliminary Study on the Relationship Among Software Metrics and Specific Vulnerability Types“. In: *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2017, S. 916–921. DOI: 10.1109/CSCI.2017.159.

[24] Isela Macia, Alessandro Garcia und Arndt Staa. „Defining and Applying Detection Strategies for Aspect-Oriented Code Smells“. In: Nov. 2010, S. 60–69. DOI: 10.1109/SBES.2010.14.

[25] Puneet Jai Kaur und Sakshi Kaushal. „Package level metrics for reusability in aspect oriented systems“. In: *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*. 2015, S. 364–368. DOI: 10.1109/ABLAZE.2015.7155021.

[26] Walter Cazzola und Alessandro Marchetto. „A concern-oriented framework for dynamic measurements“. In: *Information and Software Technology* 57 (Jan. 2014). DOI: 10.1016/j.infsof.2014.08.006.

[27] Eduardo Figueiredo u. a. „Applying and Evaluating Concern-Sensitive Design Heuristics“. In: *2009 XXIII Brazilian Symposium on Software Engineering*. 2009, S. 83–93. DOI: 10.1109/SBES.2009.14.

[28] Marco Valente u. a. „On the benefits of quantification in AspectJ systems“. In: *Journal of the Brazilian Computer Society* 16 (Aug. 2010), S. 133–146. DOI: 10.1007/s13173-010-0008-0.

[29] Freddy Munoz u. a. „Usage and testability of AOP: An empirical study of AspectJ“. In: *Information and Software Technology* 55.2 (2013). Special Section: Component-Based Software Engineering (CBSE), 2011, S. 252–266. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2012.08.003. URL: https://www.sciencedirect.com/science/article/pii/S0950584912001577.

[30] Juliana Padilha u. a. „Detecting God Methods with Concern Metrics An Exploratory Study“. In: Sep. 2013.

[31] Parthipan S, Senthil Velan S und Chitra Babu. „Design Level Metrics to Measure the Complexity Across Versions of AO Software“. In: *CoRR* abs/2012.00276 (2020). URL: https://arxiv.org/abs/2012.00276.

[32] Jorg Liebig u. a. „An analysis of the variability in forty preprocessor-based software product lines“. In: *2010 ACM/IEEE 32nd International Conference on Software Engineering*. Bd. 1. 2010, S. 105–114. DOI: 10.1145/1806799.1806819.

[33] Ramon Abilio u. a. „Detecting Code Smells in Software Product Lines-An Exploratory Study“. In: *Proceedings - 12th International Conference on Information Technology: New Generations, ITNG 2015* (Mai 2015), S. 433–438. DOI: 10.1109/ITNG.2015.76.

[34] Marcus Vinicius Couto, Marco Tulio Valente und Eduardo Figueiredo. „Extracting Software Product Lines: A Case Study Using Conditional Compilation“. In: *2011 15th European Conference on Software Maintenance and Reengineering*. 2011, S. 191–200. DOI: 10.1109/CSMR.2011.25.

[35] Ján Kohut und Valentino Vranić. „Guidelines for Using Aspects in Product Lines“. In: Jan. 2010. DOI: 10.1109/SAMI.2010.5423741.

[36] Eliane Martins. *Report: Complexity, Functional Cyclomatic and Interface*. https://www.ic.unicamp.br/~eliane/Cursos/RSM/rsm_exemplo2.htm. [Online; zuletzt zugegriffen am 20.10.2022]. 2002.

[37] Satwinder Singh, K. S. Kahlon und Parvinder S. Sandhu. „Re-engineering to analyze and measure object oriented paradigms“. In: *2010 2nd IEEE International Conference on Information Management and Engineering*. 2010, S. 472–478. DOI: 10.1109/ICIME.2010.5478013.

[38] A. Yadav und R.A. Khan. „Development of Encapsulated Class Complexity Metric“. In: *Procedia Technology* 4 (2012). 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012, S. 754–760. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2012.05.123. URL: https://www.sciencedirect.com/science/article/pii/S2212017312004021.

[39] Ruchika Malhotra und Megha Khanna. „A New Metric for Predicting Software Change Using Gene Expression Programming“. In: *Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics*. WETSoM 2014. Hyderabad, India: Association for Computing Machinery, 2014, S. 8–14. ISBN: 9781450328548. DOI: 10.1145/2593868.2593870. URL: https://doi.org/10.1145/2593868.2593870.

[40] Michele Lacchia. *Introduction to Code Metrics*. https://radon.readthedocs.io/en/latest/intro.html. [Online; zuletzt zugegriffen am 20.10.2022]. 2007.

[41] Istehad Chowdhury und Mohammad Zulkernine. „Can Complexity, Coupling, and Cohesion Metrics Be Used as Early Indicators of Vulnerabilities?“ In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. SAC '10. Sierre, Switzerland: Association for Computing Machinery, 2010, S. 1963–1969. ISBN: 9781605586397. DOI: 10.1145/1774088.1774504. URL: https://doi.org/10.1145/1774088.1774504.

[42] Ayse Tosun Mısırlı u. a. „Different Strokes for Different Folks: A Case Study on Software Metrics for Different Defect Categories“. In: *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*. WETSoM '11. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, S. 45–51. ISBN: 9781450305938. DOI: 10.1145/1985374.1985386. URL: https://doi.org/10.1145/1985374.1985386.

[43] Ezgi Erturk und Ebru Akcapinar Sezer. „A comparison of some soft computing methods for software fault prediction“. In: *Expert Systems with Applications* 42.4 (2015), S. 1872–1879. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2014.10.025. URL: https://www.sciencedirect.com/science/article/pii/S0957417414006496.

[44] D. I. De Silva u. a. „Applicability of three cognitive complexity metrics“. In: *2013 8th International Conference on Computer Science & Education*. 2013, S. 573–578. DOI: 10.1109/ICCSE.2013.6553975.

[45] Sanjay Misra u. a. „A Suite of Object Oriented Cognitive Complexity Metrics“. In: *IEEE Access* 6 (2018), S. 8782–8796. DOI: 10.1109/ACCESS.2018.2791344.

[46] Ana Luisa Medeiros u. a. „Concern-Based Assessment of Architectural Stability: A Comparative Study“. In: *2010 Fourth Brazilian Symposium on Software Components, Architectures and Reuse*. 2010, S. 130–139. DOI: 10.1109/SBCARS.2010.23.

[47] Rachel Burrows u. a. „An Empirical Evaluation of Coupling Metrics on Aspect-Oriented Programs“. In: *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*. WETSoM '10. Cape Town, South Africa: Association for Computing Machinery, 2010, S. 53–58. ISBN: 9781605589763. DOI: 10.1145/1809223.1809231. URL: https://doi.org/10.1145/1809223.1809231.

[48] A. Di Stefano u. a. „Metrics for Evaluating Concern Separation and Composition“. In: *Proceedings of the 2005 ACM Symposium on Applied Computing*. SAC '05. Santa Fe, New Mexico: Association for Computing Machinery, 2005, S. 1381–1382. ISBN: 1581139640. DOI: 10.1145/1066677.1066989. URL: https://doi.org/10.1145/1066677.1066989.

[49] Nelio Cacho u. a. „Blending design patterns with aspects: A quantitative study“. In: *Journal of Systems and Software* 98 (2014), S. 117–139. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2014.08.041. URL: https://www.sciencedirect.com/science/article/pii/S0164121214001885.

[50] Andrzej Olszak und Bo Nørregaard Jørgensen. „Remodularizing Java Programs for Improved Locality of Feature Implementations in Source Code“. In: *Sci. Comput. Program.* 77.3 (März 2012), S. 131–151. ISSN: 0167-6423. DOI: 10.1016/j.scico.2010.10.007. URL: https://doi.org/10.1016/j.scico.2010.10.007.

[51] María Laura Ponisio und Oscar Nierstrasz. „Using Contextual Information to Assess Package Cohesion“. In: 2006.

[52] Sonika Jindal und Gauri Khurana. „The statistical analysis of source-code to determine the refactoring opportunities factor (ROF) using a machine learning algorithm“. In: *Fifth International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom 2013)*. 2013, S. 396–403. DOI: 10.1049/cp.2013.2244.

[53] Daniele Romano und Martin Pinzger. „Using source code metrics to predict change-prone Java interfaces“. In: *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. 2011, S. 303–312. DOI: 10.1109/ICSM.2011.6080797.

[54] Jehad Al Dallal. „Improving the applicability of object-oriented class cohesion metrics“. In: *Information and Software Technology* 53.9 (2011). Studying work practices in Global Software Engineering, S. 914–928. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2011.03.004. URL: https://www.sciencedirect.com/science/article/pii/S0950584911000632.

[55] Jehad Al Dallal. „Transitive-based object-oriented lack-of-cohesion metric“. In: *Procedia Computer Science* 3 (2011). World Conference on Information Technology, S. 1581–1587. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2011.01.053. URL: https://www.sciencedirect.com/science/article/pii/S1877050911000548.

[56] Yuming Zhou u. a. „An In-Depth Study of the Potentially Confounding Effect of Class Size in Fault Prediction“. In: *ACM Trans. Softw. Eng. Methodol.* 23.1 (Feb. 2014). ISSN: 1049-331X. DOI: 10.1145/2556777. URL: https://doi.org/10.1145/2556777.

[57] Jehad Al Dallal. „Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics“. In: *Information and Software Technology* 54.10 (2012), S. 1125–1141. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2012.04.004. URL: https://www.sciencedirect.com/science/article/pii/S0950584912000754.

[58] Kecia Aline Marques Ferreira u. a. „The evolving structures of software systems“. In: *2012 3rd International Workshop on Emerging Trends in Software Metrics (WETSoM)*. 2012, S. 28–34. DOI: 10.1109/WETSoM.2012.6226989.

[59] Yu Qu u. a. „Exploring community structure of software Call Graph and its applications in class cohesion measurement“. In: *Journal of Systems and Software* 108 (2015), S. 193–210. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2015.06.015. URL: https://www.sciencedirect.com/science/article/pii/S0164121215001259.

[60] José M. Conejero u. a. „Mining early aspects based on syntactical and dependency analyses“. In: *Science of Computer Programming* 75.11 (2010). Special Section on the Programming Languages Track at the 23rd ACM Symposium on Applied Computing, S. 1113–1141. ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2010.04.013. URL: https://www.sciencedirect.com/science/article/pii/S0167642310000821.

[61] Sven Apel und Dirk Beyer. „Feature Cohesion in Software Product Lines: An Exploratory Study“. In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, S. 421–430. ISBN: 9781450304450. DOI: 10.1145/1985793.1985851. URL: https://doi.org/10.1145/1985793.1985851.

[62] Heliomar Santos u. a. „Software rejuvenation via a multi-agent approach“. In: *Journal of Systems and Software* 104 (2015), S. 41–59. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2015.02.017. URL: https://www.sciencedirect.com/science/article/pii/S0164121215000412.

[63] V. Krishnapriya und K. Ramar. „Exploring the Difference Between Object Oriented Class Inheritance and Interfaces Using Coupling Measures“. In: *2010 International Conference on Advances in Computer Engineering*. 2010, S. 207–211. DOI: 10.1109/ACE.2010.25.

[64] Francesca Arcelli Fontana u. a. „Automatic Metric Thresholds Derivation for Code Smell Detection“. In: *2015 IEEE/ACM 6th International Workshop on Emerging Trends in Software Metrics*. 2015, S. 44–53. DOI: 10.1109/WETSoM.2015.14.

[65] Sonika Jindal und Gauri Khurana. „The statistical analysis of source-code to determine the refactoring opportunities factor (ROF) using a machine learning algorithm“. In: *Fifth International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom 2013)*. 2013, S. 396–403. DOI: 10.1049/cp.2013.2244.

[66] Claudio Sant'Anna u. a. „On the reuse and maintenance of aspect-oriented software: An assessment framework“. In: *Brazilian Symposium on Software Engineering* (Jan. 2003).

[67] Avadhesh Kumar, Rajesh Kumar und P. S. Grover. „Generalized Coupling Measure for Aspect-Oriented Systems“. In: *SIGSOFT Softw. Eng. Notes* 34.3 (Mai 2009), S. 1–6. ISSN: 0163-5948. DOI: 10.1145/1527202.1527209. URL: https://doi.org/10.1145/1527202.1527209.

[68] Mario Luca Bernardi und Giuseppe A. Di Lucca. „A Metric Model for Aspects' Coupling“. In: *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*. WETSoM '10. Cape Town, South Africa: Association for Computing Machinery, 2010, S. 59–66. ISBN: 9781605589763. DOI: 10.1145/1809223.1809232. URL: https://doi.org/10.1145/1809223.1809232.

[69] Aivosto Oy. *MOOD and MOOD2 metrics*. https://www.aivosto.com/. [Online; zuletzt zugegriffen am 20.10.2022]. 2006.

[70] Sandeep Kaur, Kanwaljeet Kaur und Navjot Kaur. „An Empirical Investigation of Relationship between Software Metrics“. In: *2015 Second International Conference on Advances in Computing and Communication Engineering*. 2015, S. 639–643. DOI: 10.1109/ICACCE.2015.23.

[71] Gagandeep Makkar, Jitender Kumar Chhabra und Rama Krishna Challa. „Object oriented inheritance metric-reusability perspective“. In: *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*. 2012, S. 852–859. DOI: 10.1109/ICCEET.2012.6203815.

[72] G. Sri Krishna und Rushikesh K. Joshi. „Inheritance Metrics: What Do They Measure?“ In: *Proceedings of the 4th Workshop on MechAnisms for SPEcialization, Generalization and InHerItance*. MASPEGHI '10. Maribor, Slovenia: Association for Computing Machinery, 2010. ISBN: 9781450305358. DOI: 10.1145/1929999.1930000. URL: https://doi.org/10.1145/1929999.1930000.

[73] Vinobha A, Senthil Velan S und Chitra Babu. „Evaluation of reusability in Aspect Oriented Software using inheritance metrics“. In: *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*. IEEE, Mai 2014. DOI: 10.1109/icaccct.2014.7019401. URL: https://doi.org/10.1109%2Ficaccct.2014.7019401.

[74] Adauto Almeida u. a. „Is Exception Handling a Reusable Aspect?“ In: *2014 Eighth Brazilian Symposium on Software Components, Architectures and Reuse*. 2014, S. 32–41. DOI: 10.1109/SBCARS.2014.17.

[75] V.S.P. Srivastav und Piyush Prakash. „Green metrics for OO codes: CAAEC metric“. In: *2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)*. 2013, S. 296–298. DOI: 10.1109/ICGCE.2013.6823448.

[76] Giulio Concas u. a. „Assessing Traditional and New Metrics for Object-Oriented Systems“. In: *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*. WETSoM '10. Cape Town, South Africa: Association for Computing Machinery, 2010, S. 24–31. ISBN: 9781605589763. DOI: 10.1145/1809223.1809227. URL: https://doi.org/10.1145/1809223.1809227.

[77] Yutao Ma u. a. „A Hybrid Set of Complexity Metrics for Large-Scale Object-Oriented Software Systems“. In: *Journal of Computer Science and Technology* 25 (2010), S. 1184–1201.

[78] Yutao Ma u. a. „How multiple-dependency structure of classes affects their functions a statistical perspective“. In: *2010 2nd International Conference on Software Technology and Engineering*. Bd. 2. 2010, S. V2-60-V2–66. DOI: 10.1109/ICSTE.2010.5608763.

[79] Mitsuhiro Nakamura und Tomoki Hamagami. „A Software Quality Evaluation Method Using the Change of Source Code Metrics“. In: *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*. 2012, S. 65–69. DOI: 10.1109/ISSREW.2012.13.

6 Tabellenverzeichnis

7 Anlagen

7.1 Anlage A - Quantifizierungsmetriken

7.1.1 Anlage A.1 - Allgemeine Quantifizierungsmetriken

Metrik	**Beschreibung**
DSC [1]	*Designgröße in Klassen (Design Size in Classes)*: Gesamte Anzahl aller Klassen einer Design-Spezifikation [17]
PACK [16]	*Anzahl importierter Pakete (Number of Imported Packages)*: Anzahl der durch eine Klasse oder das gesamte System importierten Pakete [16]
NINT, NOI, INTR [1]	*Anzahl an Schnittstellen (Number of Interfaces)*: Anzahl aller implementierten Schnittstellen [16]
NCLASS, NC [1] [2]	*Anzahl an Klassen (Number of Classes)*: Anzahl aller Klassen eines Systems oder einer Komponente
NCONS, CONS [1]	*Anzahl an Konstruktoren (Number of Constructors)*: Anzahl aller Konstruktoren eines Systems oder einer Komponente
NPAR, PAR [1]	*Parameter (Parameters)*: Anzahl an Parametern innerhalb der Deklaration einer Methode [18]
CDCF [2]	*Anzahl deklarierter Funktionen (CountDeclFunction)*: Anzahl aller deklarierten Funktionen [2]
CS [2]	*Anzahl Semikolons (CountSemicolon)*: Anzahl aller Semikolons [2]
NMS [1]	*Anzahl gesendeter Nachrichten (Number of Message Sends)*: Anzahl aller durch eine Methode gesendeten Nachrichten [19]
COOP [1]	*Aufrufe in objekt-orientierter Programmierung (Calls in Object Oriented Programming)*: Anzahl aller Aufrufe von gespeicherten Prozeduren, Methodenaufrufen innerhalb und außerhalb von Klassen sowie von Konstruktoren [20]

Tabelle 4: Allgemeine Quantifizierungsmetriken

7.1.2 Anlage A.2 - Zeilenbasierte Quantifizierungsmetriken

Metrik	**Beschreibung**
LOC, NLOC [1] [2]	*Anzahl Code-Zeilen (Lines of Code)*: Anzahl aller Code-Zeilen [21]
BLOC [1] [2]	*Anzahl leerer Code-Zeilen (Blank Lines of Code)*: Anzahl aller leeren Code-Zeilen [21]
MLOC [1]	*Anzahl Code-Zeilen einer Methode (Method Lines of Code)*: Anzahl aller Code-Zeilen einer Methode [21]
SLOC [1] [2] [22] [23]	*Anzahl Quellcode-Zeilen (Source Lines of Code)*: Anzahl aller Zeilen des Quellcodes ausschließlich Kommentare [22]
ELOC [1] [2]	*Anzahl ausführbarer Quellcode-Anweisungen (Number of Executable Source Code Statements)*: Anzahl aller ausführbaren Zeilen des Quellcodes [21]

Tabelle 4 – Fortsetzung zeilenbasierter Quantifizierungsmetriken

Metrik	**Beschreibung**
CLCD [2]	*Anzahl deklarativer Code-Zeilen (CountLineCodeDecl)*: Anzahl aller deklarativen Zeilen des Quellcodes [2]
LCOMM, CCML [1] [2]	*Anzahl Kommentarzeilen (CountLineComment)*: Anzahl aller Kommentarzeilen des Quellcodes [2]
LOCAC [2]	*Anzahl Zeilen mit Kommentaren oder Quellcode (Lines with Comments or Code)*: Anzahl aller Zeilen des Quellcodes inklusive aller Kommentarzeilen [2]
AL [2]	*Durchschnittliche Anzahl an Code-Zeilen (AvgLine)*: Durchschnittliche Anzahl an Code-Zeilen aller verschachtelten Funktionen und Methoden [2]
ALB [2]	*Durchschnittliche Anzahl leerer Zeilen (AvgLineBlank)*: Durchschnittliche Anzahl leerer Code-Zeilen aller verschachtelten Funktionen und Methoden [2]
ALC [2]	*Durchschnittliche Anzahl ausführbarer Quellcode-Zeilen (AvgLineCodes)*: Durchschnittliche Anzahl an ausführbaren Quellcode-Zeilen aller verschachtelten Funktionen und Methoden [2]
ALCOMM [2]	*Durchschnittliche Anzahl Kommentarzeilen (AvgLineComment)*: Durchschnittliche Anzahl an Kommentarzeilen aller verschachtelten Funktionen und Methoden [2]
CLI [2]	*Anzahl inaktiver Zeilen (CountLineInactive)*: Anzahl inaktiver Code-Zeilen [2]
CLP [2]	*Anzahl Zeilen zur Vorverarbeitung (CountLinePreprocessor)*: Anzahl an Code-Zeilen zur Vorverarbeitung des Quellcodes [2]
RCC [2]	*Verhältnis Kommentare zu Code (Ratio Comment To Code)*: Anzahl an Kommentarzeilen im Verhältnis zur Anzahl an der Quellcode-Zeilen [18]

Tabelle 5: Zeilenbasierte Quantifizierungsmetriken

7.1.3 Anlage A.3 - Aspekt-orientierte Quantifizierungsmetriken

Metrik	**Beschreibung**
NA [1]	*Anzahl an Aspekten (Number of Aspects)*: Anzahl aller Aspekte eines Systmes
AS [1]	*Aspektgröße (Aspect Size)*: Anzahl an Attributen, Methoden, Pointcuts, Advices und Deklarationen als Bestandteile der Implementierung eines Aspekts [24]
ADS [1]	*Designgröße eines Aspekts (Aspect Design Size)*: Anzahl an Klassen und Schnittstellen innerhalb eines Aspekts (Pakets) [25]
Size [1]	*Größe (Size)*: Anzahl der Attribute und Methoden einer Klasse mit Assoziation zu einer gegebenen Eigenschaft [26]
Spread [1]	*Verbreitung (Spread)*: Anzahl der Klassen mit Beziehungen zu einer gegebenen Eigenschaft [26]
NOCA [1]	*Anzahl der Concern-Attribute (Number of Concern Attributes)*: Anzahl verwendeter Attributen zur Realisierung eines bestimmten Concerns [27]

Tabelle 5 – Fortsetzung aspekt-orientierter Quantifizierungsmetriken

Metrik	**Beschreibung**
NOCO [1]	*Anzahl der Concern-Operationen (Number of Concern Operations)*: Anzahl verwendeter Methoden, Konstruktoren und Advices zur Realisierung eines bestimmten Concerns [27]
SR [1]	*Streuungsreduktion (Scattering Reduction)*: Anzahl verwendeter Advices zur Implementierung eines Concerns subtrahiert von der Anzahl der dazu benötigten Join-Points [28]
QD [1]	*Quantifizierungsgrad (Quantification Degree)*: *SR* im Verhältnis zur Anzahl verwendeter Join-Points [28]
NOAD [1]	*Anzahl der Advices (Number of Advices)*: Anzahl aller Advices mit mindestens einem Joinpoint [29]
NOIAD [1]	*Anzahl der Advices zur Realisierung invasiver Muster (Number of Advices Realizing Invasiveness Patterns)*: Anzahl aller Advices mit mindestens einem Joinpoint und der Realisierung eines oder mehrerer Invasivitätsmuster [29]
NARI [1]	*Anzahl der Advices zur Realisierung einzelner invasiver Muster (Number of Advices Realizing each Invasiveness pattern)*: Anzahl aller Advices, welche ein gegebenes Invasivitätsmuster realisieren [29]
AMR [1]	*Verhältnis der Advices pro Methode (Advices per Method Ratio)*: Anzahl aller Advices mit mindestens einem Joinpoint im Verhältnis zur Anzahl aller Methoden [29]
IAMR [1]	*Verhältnis invasiver Advices pro Methdoe (Invasive Advices per Method Ratio)*: Anzahl aller Advices mit mindestens einem Joinpoint und der Realisierung eines oder mehrerer Invasivitätsmuster im Verhältnis zur Anzahl aller Methoden [29]
NAJP [1]	*Anzahl für Advice verwendete Joinpoints (Number of Join Points Advised by an Advice)*: Anzahl aller durch einen Advice verwendeten Joinpoints [29]
NASJP [1]	*Anzahl an Aspekten mit Bezug zum gleichen Joinpoint (Number of Aspects Referring to Shared Join Point)*: Anzahl an Aspekten mit der gleichen Auswahl an Joinpoints [24]
CIJP [1]	*(Cumulated Number of Join Points Matched by the Invasive Advices)*: Anzahl aller für Advices verwendete Joinpoints [29]
NCC [1]	*Anzahl der Concerns pro Komponente (Number of Concerns per Component)*: Anzahl aller implementierten Concerns im Verhältnis zur gesamten Anzahl aller Klassen und Aspekte [27]
NOCM [1]	*Anzahl der Concerns pro Methode (Number of Concerns per Method)*: Anzahl der durch jede Methode einer Klasse implementierten Concerns [30]
LOCC [1]	*Code-Zeilen eines Concerns (Concern Lines of Code)*: Anzahl an Code-Zeilen zur Implementierung eines Concerns [30]
NOAC [1]	*Anzahl der Attribute pro Klasse (Number of Attributes per Class)*: Anzahl der Attribute aller Klassen im Verhältnis zur gesamten Anzahl an Klassen [31]

Tabelle 6: Aspekt-orientierte Quantifizierungsmetriken

7.1.4 Anlage A.4 - Merkmals-orientierte Quantifizierungsmetriken

Metrik	**Beschreibung**
GRAN [1]	*Granularität (Granularity)*: Anzahl an Annotationen auf sechs verschiedenen Ebenen (Global, Funktionen und Datentypen, Blöcke, Aussagen und Ausdrücken sowie Funktionssignaturen)
NOFC [1]	*Anzahl an Merkmalskonstanten (Number of Feature Constants)*: Summe aller Merkmalskonstanten aus Merkmalsausdrücken eines Projekts [32]
LOF [1]	*Anzahl Code-Zeilen eines Merkmals (Lines of Feature Code)*: Anzahl von Code-Zeilen aller Merkmals-Annotationen [32]
NOCT [1]	*Anzahl an Konstanten (Number of Constants)*: Anzahl der Konstanten (Klassen und Schnittstellen) einer Softwareproduktlinie [33]
NOF [1]	*Anzahl an Merkmalen (Number of Features)*: Anzahl der Merkmale einer Softwareproduktlinie [33]
NOC [1]	*Anzahl an Komponenten (Number of Components)*: Anzahl der Komponenten einer Softwareproduktlinie [33]
NOR [1]	*Anzahl an Verfeinerungen (Number of Refinements)*: Anzahl der Verfeinerungen (Klassen und Schnittstellen) einer Softwareproduktlinie [33]
NCR [1]	*Anzahl an Konstanten-Verfeinerungen (Number of Constant Refinements)*: Anzahl der Verfeinerungen einer Konstante [33]
NMR [1]	*Anzahl an Methoden-Verfeinerungen (Number of Method Refinements)*: Anzahl der Verfeinerungen einer Methode [33]
NRC [1]	*Anzahl verfeinerter Konstanten (Number of Refined Constants)*: Anzahl der verfeinerter Konstanten einer Softwareproduktlinie [33]
NRM [1]	*Anzahl verfeinerter Methoden (Number of Refined Methods)*: Anzahl der verfeinerter Methoden einer Softwareproduktlinie [33]
NAN [1]	*Anzahl an Annotationen (Number of Annotations)*: Anzahl der Annotationen einer Softwareproduktlinie [33]
NFCS [1]	*Anzahl annotierter Code-Teile mit Verbindung zu einer Klassensignatur (Number of Pieces of Annotated Code Associated to a Class Signature)*: Anzahl der zu einer Klassensignatur zugehörigen annotierten Code-Teile [34]
NFP [1]	*Anzahl vollständig annotierter Pakete zur Implementierung eines Merkmals (Number of Packages Entirely Annotated as Implementing a Feature)*: Anzahl der zur Implementierung eines Merkmals vollständig annotierten Pakete [34]
NFC [1]	*Anzahl vollständig annotierter Klassen zur Implementierung eines Merkmals (Number of Classes Entirely Annotated as Implementing a Feature)*: Anzahl der zur Implementierung eines Merkmals vollständig annotierten Klassen [34]
NFIM [1]	*Anzahl annotierter Methodensignaturen zur Implementierung eines Merkmals (Number of Method Signatures in an Annotated Interface)*: Anzahl der zur Implementierung eines Merkmals vollständig annotierten Methodensignaturen innerhalb von Schnittstellen [34]

Tabelle 6 – Fortsetzung merkmals-orientierter Metriken

Metrik	**Beschreibung**
NFM [1]	*Anzahl vollständig annotierter Methoden zur Implementierung eines Merkmals (Number of Methods Entirely Annotated as Implementing a Feature)*: Anzahl der zur Implementierung eines Merkmals vollständig annotierten Methoden [34]
NFMB [1]	*Anzahl vollständig annotierter Methodenkörper zur Implementierung eines Merkmals (Number of Methods Bodies Entirely Annotated as Implementing a Feature)*: Anzahl der zur Implementierung eines Merkmals vollständig annotierten Methodenkörper [34]
NFA [1]	*Anzahl annotierter Attribute zur Implementierung eines Merkmals (Number of Attributes Annotated to Implement a Feature)*: Anzahl der zur Implementierung eines Merkmals annotierten Klassen- und Instanzattribute [34]
NFS [1]	*Anzahl annotierter Anweisungen zur Implementierung eines Merkmals (Number of Statements Annotated to Implement a Feature)*: Anzahl der zur Implementierung eines Merkmals annotierten Aussagen inklusive Methodenaufrufe, Zuweisungen, Bedingungen, Schleifen, etc. [34]
NFE [1]	*Anzahl annotierter Ausdrücke zur Implementierung eines Merkmals (Number of Expressions Annotated to Implement a Feature)*: Anzahl der zur Implementierung eines Merkmals annotierten Ausdrücke wie beispielsweise die Ausdrücke innerhalb von Bedingungen oder Schleifen [34]
NFMC [1]	*Anzahl an Methoden mit annotierten Code (Number of Methods with Annotated Code)*: Anzahl an Methoden mit annotierten Quellcode
NAC [1]	*Anzahl betroffener Klassen (Number of Affected Classes)*: Anzahl aller zur Implementierung eines Merkmals betroffene Klassen [35]

Tabelle 7: Merkmals-orientierte Quantifizierungsmetriken

7.2 Anlage B - Komplexitätsmetriken

7.2.1 Anlage B.1 - Allgemeine Komplexitätsmetriken

Metrik	**Beschreibung**
INC [1]	*Schnittstellenkomplexität (Interface Complexity)*: Anzahl aller Eingabeparameter einer Funktion inklusive der Anzahl aller Rückgabezustände dieser Funktion [36]
WCCF [1]	*Gewichteter Klassenkomplexitätsfaktor (Weighted Class Complexity Factor)*: Anzahl der Methoden und Attribute einer Klasse im Verhältnis zur Anzahl der Attribute und Methoden aller Klassen [37]
HK [1]	*Henry-Kafury-Größe (Henry Kafura Size)*: $(CA * CE)^2$ [4]
EC2M [1]	*Gekapselte Klassenkomplexitätsmetrik (Encapsulated Class Complexity Metric)*: Summe der Anzahl aller Methoden und Attribute mit ausschließlicher Verwendung innerhalb der eigenen Klasse sowie die Summe der Anzahl an Zugriffen von Methoden und Attributen außerhalb der eigenen Klassen im Verhältnis zur Anzahl an Methoden und Attributen aller Klassen [38]
CI [1]	*Änderungsindex (Change Index)*: 0.24 * *RFC* + 2 * *SLOC* - *WMC* - 59.62 - *LCOM1* als Indikator der Anfälligkeit von Veränderungen einzelner Klassen [39]
SEIMI, MI SEI [1]	*Wartbarkeitsindex (Maintainability index)*: Messung der Wartbarkeit von Quellcode auf Grundlage verschiedener existierender Formeln als Kombination aus den Metriken *VOL*, *VG*, *SLOC* und *RCC* [40]

Tabelle 8: Allgemeine Komplexitätsmetriken

7.2.2 Anlage B.2 - Zyklomatische Komplexitätsmetriken

Metrik	**Beschreibung**
VG, CYCLO [1] [2] [16]	*McCabe's zyklomatische Komplexität (McCabe Cyclomatic Complexity)*: Anzahl der unabhängigen Pfade durch eine Programmeinheit wie beispielsweise die Anzahl aller Entscheidungsanweisungen addiert mit 1 [41]
MVG [1] [2] [22]	*Modifizierte zyklomatische Komplexität (Modified Cyclomatic Complexity)*: Identisch zu *VG* mit dem einzigen Unterschied, dass die Case-Blöcke einer Switch-Anweisung nicht einzeln gezählt werden, sondern die gesamte Switch-Anweisung als 1 gezählt wird [41]
SVG, CCS [1] [2]	*Strikte zyklomatische Komplexität (Strict Cyclomatic Complexity)*: Identisch zu *VG* mit dem einzigen Unterschied, dass logische Operatoren wie *AND* und *OR* ebenfalls gezählt werden [41]
EVG, CCE [1] [2] [22]	*Essentielle zyklomatische Komplexität (Essential Cyclomatic Complexity)*: Berechnung identisch zu dem Prinzip von *VG*, nachdem alle strukturierten Programmierprimitiven iterativ durch einzelne Anweisungen ersetzt wurden [41]

Tabelle 8 – Fortsetzung zyklomatischer Komplexitätsmetriken

Metrik	**Beschreibung**
NVG [1]	*Normalisierte zyklomatische Komplexität (Normalized Cyclomatic Complexity)*: Zyklomatische Komplexität (*VG*) im Verhältnis zur Anzahl unterschiedlicher Operatoren (*NDO1*) [42]
IVG [1] [2]	*Designkomplexität (Design Complexity)*: Zyklomatische Komplexität des reduzierten Flussdiagramms eines Moduls ohne Entscheidungen und Schleifen mit Aufrufen von Untermodulen [43]
AVG [2]	*Durchschnittliche zyklomatische Komplexität (Average Cyclomatic Complexity)*: Durchschnittliche zyklomatische Komplexität aller verschachtelten Funktionen und Methoden [2]
AMVG [2]	*Durchschnittliche modifizierte zyklomatische Komplexität (Average Modified Cylcomatic Complexity)*: Durchschnittliche modifizierte zyklomatische Komplexität aller verschachtelten Funktionen und Methoden [2]
ASVG [2]	*Durchschnittliche strikte zyklomatische Komplexität (Average Strict Cyclomatic Complexity)*: Durchschnittliche strikte zyklomatische Komplexität aller verschachtelten Funktionen und Methoden [2]
AEVG [2]	*Durchschnittliche essentielle zyklomatische Komplexität (Average Essential Cyclomatic Complexity)*: Durchschnittliche essentielle zyklomatische Komplexität aller verschachtelten Funktionen und Methoden [2]
MAXVG [2]	*Maximale zyklomatische Komplexität (Maximum Cyclomatic Complexity)*: Maximale zyklomatische Komplexität aller verschachtelten Funktionen und Methoden [2]
MMVG [2]	*Maximale modifizierte zyklomatische Komplexität (Maximum Modified Cyclomatic Complexity)*: Maximale modifizierte zyklomatische Komplexität aller verschachtelten Funktionen und Methoden [2]
MSVG [2]	*Maximale strikte zyklomatische Komplexität (Maximum Strict Cyclomatic Complexity)*: Maximale strikte zyklomatische Komplexität aller verschachtelten Funktionen und Methoden [2]
WMC, SUMVG [1] [2]	*Gewichtete Methoden pro Klasse (Weighted Methods per Class)*: Summe der zyklomatischen Komplexität aller verschachtelten Funktionen und Methoden [2]
SMVG [2]	*Summe modifizierter zyklomatischer Komplexitäten (Sum of Modified Cyclomatic Complexity)*: Summe der modifizierten zyklomatischen Komplexität aller verschachtelten Funktionen und Methoden [2]
SSVG [2]	*Summe strikter zyklomatischer Komplexitäten (Sum of Strict Cyclomatic Complexity)*: Summe der strikten zyklomatischen Komplexität aller verschachtelten Funktionen und Methoden [2]
SEVG [2]	*Summe essentieller zyklomatischer Komplexitäten (Sum of Essential Cyclomatic Complexity)*: Summe der essentiellen zyklomatischen Komplexität aller verschachtelten Funktionen und Methoden [2]
SDMC [1]	*Standardabweichung der Methodenkomplexität (Standard Deviation Method Complexity)*: Quadratische Wurzel der Varianz zyklomatischer Komplexität von in einer Klasse implementierten Methoden [5]

Tabelle 8 – Fortsetzung zyklomatischer Komplexitätsmetriken

Metrik	**Beschreibung**
NEST, BD, NBD [1]	*Verschachtelungsebene (Nesting Level)*: Maximale Verschachtelungstiefe einer Kontrollstruktur
MN, MAXNEST [2] [22]	*Maximale Verschachtelungsebene (Maximum Nesting)*: Maximale Verschachtelungstiefe aller Kontrollstrukturen [18]
BC [2]	*Verzweigungsanzahl (Branch Count)*: Anzahl an Verzweigungen im Kontrollfluss eines gegebenen Moduls [15]
KNOTS [2]	*Anzahl an Knoten (Number of Knots)*: Anzahl überlappender Sprünge im Kontrollfluss aufgrund von bedingten Entscheidungen und Schleifenwiederholungen [2]
MAXEK [2]	*Maximale Anzahl essentieller Knoten (MaxEssentialKnots)*: Maximale Anzahl überlappender Sprünge im Kontrollfluss nach der Beseitigung von Konstrukten struktureller Programmierung [2]
MINEK [2]	*Minimale Anzahl essentieller Knoten (MinEssentialKnots)*: Minimale Anzahl überlappender Sprünge im Kontrollfluss nach der Beseitigung von Konstrukten struktureller Programmierung [2]
NPATH [1]	*Anzahl möglicher Pfade (Number of Possible Paths)*: Anzahl möglicher, eindeutiger Entscheidungspfade ohne abnormale Abbrüche oder Sprünge [2] [22]

Tabelle 9: Zyklomatische Komplexitätsmetriken

7.2.3 Anlage B.3 - Halstead-Komplexitätsmetriken

Metrik	**Beschreibung**
NDO1 [1] [2]	*n1*: Anzahl unterschiedlicher Operatoren nach Halstead [6]
TNO1 [1] [2]	*N1*: Anzahl aller Operatoren nach Halstead [6]
NDO2 [1] [2]	*n2*: Anzahl unterschiedlicher Operanden nach Halstead [6]
TNO2 [1] [2]	*N2*: Anzahl aller Operanden nach Halstead [6]
PV [1]	*Programmvokabular (Program Vocabulary)*: Anzahl aller unterschiedlichen Operatoren (*NDO1*) und aller unterschiedlichen Operatoren (*NDO2*) [6]
PL [1]	*Programmlänge (Program Length)*: Anzahl aller Operatoren (*TNO1*) und aller Operanden (*TNO2*) [6]
VOL [1] [2]	*Halstead-Volumen (Halstead Volume)*: $PL * \log_2(PV)$ als Repräsentation des Umfangs zur Kodierung des Programms in einem Alphabet [6]
HSD [2]	*Halstead-Schwierigkeit (Halstead Difficulty)*: $(NDO1/2)/(TNO2/NDO2)$ als Repräsentation der Schwierigkeit bezüglich des Umgangs mit dem Programm [15]
HSE [2]	*Halstead-Aufwand (Halstead Effort)*: $VOL * HSD$ als Repräsentation zur Abschätzung des Aufwands mentaler Aktivitäten zur Implementierung eines Algorithmus als Programm in einer spezifischen Programmiersprache [15]

Tabelle 9 – Fortsetzung der Halstead-Komplexitätsmetriken

Metrik	Beschreibung
HSPT [2]	*Halstead-Programmierzeit (Halstead Programming Time)*: $HSE/18$ als Repräsentation der Zeit in Minuten zur Implementierung eines Algorithmus als Programm in einer spezifischen Programmiersprache [15]
HSEE [2]	*Halstead-Fehlerabschätzung (Halstead Error Estimate)*: $(E^{2/3})/3000$ als Repräsentation der geschätzten Anzahl an Fehlern nach Implementierung des Programms [2]

Tabelle 10: Halstead-Komplexitätsmetriken

7.2.4 Anlage B.4 - Kognitive Komplexitätsmetriken

Metrik	Beschreibung
CWCM [1]	*Komplexitätsmaß des kognitiven Gewichts (Cognitive Weight Complexity Measure)*: Summe der kognitiven Gewichte der Basiskontrollstrukturen einer Komponente [44]
CICM [1]	*Komplexitätsmaß der kognitiven Information (Cognitive Information Complexity Measure)*: Summe der kognitiven Gewichte der Basiskontrollstrukturen einer Komponente multipliziert mit dem gewichteten Informationsgehalt der Komponente [44]
CFS [1]	*Kognitive funktionale Größe (Cognitive Functional Size)*: Summe der kognitiven Gewichte von Basiskontrollstrukturen einer Methode multipliziert mit der Anzahl aller Ein- und Ausgabeparameter und für Klassen bzw. Komponenten die Summe der CFS aller darin enthaltenen Methoden *Cognitive Functional Size)*: Summe der kognitiven Gewichte von Basiskontrollstrukturen einer Methode multipliziert mit der Anzahl aller Ein- und Ausgabeparameter und für Klassen bzw. Komponenten die Summe der CFS aller darin enthaltenen Methoden
MC [1]	*Methodenkomplexität (Method Complexity)*: Summe der kognitiven Gewichte aller Basiskontrollstrukturen der Methoden einer Komponente inklusive der kognitiven Gewichte der darin aufgerufenen Methoden [45]
AC [1]	*Attributkomplexität (Attribute Complexity)*: Anzahl aller einer Klasse oder Komponente zugewiesenen Attribute [45]
WCC [1]	*Gewichtete Klassenkomplexität (Weighted Class Complexity)*: Summe der kognitiven Gewichte aller Attribute (AC) und Methoden (MC) einer Klasse [45]
COCI [1]	*Codekomplexität (Code Complexity)*: Kognitive Komplexität des gesamten Systems durch die Multiplikation der kognitiven Gewichte von Klassen mit Vererbungsbeziehungen und der Summe aller unabhängigen Vererbungshierarchien [45]
AMC [1]	*Durchschnittliche Methodenkomplexität (Average Method Complexity)*: Summe der kognitiven Komplexität aller Methoden einer Klasse im Verhältnis zur Anzahl aller Methoden der Klasse [45]

Tabelle 10 – Fortsetzung kognitiver Komplexitätsmetriken

Metrik	Beschreibung
AMCC [1]	*Durchschnittliche Methodenkomplexität pro Klasse (Average Method Complexity per Class)*: Summe der durchschnittlichen, kognitiven Methodenkomplexität aller Klassen (AMC) im Verhältnis zur Anzahl aller Klassen [45]
ACC [1]	*Durchschnittliche Klassenkomplexität (Average Class Complexity)*: Summe der kognitiven Klassenkomplexität aller Klassen (WCC) im Verhältnis zur Anzahl aller Klassen [45]
AAC [1]	*Durchschnittliche Anzahl an Attributen pro Klasse (Average Attributes per Class)*: Summe der kognitiven Attributkomplexität (AAC) aller Klassen im Verhältnis zur Anzahl aller Klassen [45]

Tabelle 11: Kognitive Komplexitätsmetriken

7.2.5 Anlage B.5 - Aspekt-orientierte Komplexitätsmetriken

Metrik	Beschreibung
CONC [1]	*Konzentration (Concentration)*: Anzahl an Methoden einer Komponente zur Realisierung eines Concerns im Verhältnis zur gesamten Anzahl an Methoden zur Realisierung eines Concerns [46]
DEDIC [1]	*Zueignung (Dedication)*: Anzahl an Methoden einer Komponente zur Realisierung eines Concerns im Verhältnis zur gesamten Anzahl an Methoden der Komponente [46]
WOM [1]	*Gewichtete Operationen innerhalb eines Moduls (Weighted Operations in a Module)*: Anzahl an Operationen, Advices und Introductions einer Klasse oder Aspekts gewichtet auf Grundlage der jeweiligen internen Komplexität [47]
CDLOC [1]	*Zerstreuung eines Concerns über Code-Zeilen (Concern Diffusion over Lines of Code)*: Anzahl der Übergangspunkte eines Concerns zu einem anderen Concern im Verhältnis zur Code-Zeilen des Concerns [30]
FRAGM [1]	*Fragmentierung (Fragmentation)*: Berechnung der Heterogenität von Concerns einer Klasse auf Grundlage von Methodenaufrufen [48]
WOC [1]	*Gewichtete Operationen pro Komponente (Weighted Operations per Component)*: Summe der Anzahl an Methoden und Advices und ihrer Parameter aller Klassen und Aspekte [49]
WAA [1]	*Gewichtete Advices pro Klasse (Weighted Advices per Class)*: Summe der kognitiven Gewichte aller Advice-Typen eines gegebenen Aspekts [31]
WJPA [1]	*Gewichtete Joinpoints pro Aspekt(Weighted Join Points per Aspect)*: Summe der kognitiven Gewichte von Joinpoints in Klassen und Aspekten mit Beziehungen zu einem gegebenen Aspekt [31]
WMCA [1]	*Gewichtete Methoden pro Klasse und Aspekt (Weighted Methods per Class and Aspect)*: Summe der kognitivien Gewichte von Methoden einer Klasse oder eines Aspekts [31]

Tabelle 11 – Fortsetzung aspekt-orientierter Komplexitätsmetriken

Metrik	**Beschreibung**
WPA [1]	*Gewichtete Pointcuts pro Aspekt (Weighted Pointcut per Aspect)*: Summe der kognitiven Gewichte aller Pointcut-Designatoren und Joinpoint-Signaturen eines gegebenen Aspekts [31]
SPP [1]	*Menge primitiver Pointcuts (Set of Primitive Pointcuts)*: Anzahl der in einem Pointcut-Ausdruck verwendeten primitiven Pointcuts [24]

Tabelle 12: Aspekt-orientierte Komplexitätsmetriken

7.2.6 Anlage B.6 - Merkmals-orientierte Komplexitätsmetriken

Metrik	**Beschreibung**
FSCA [1]	*Streuung eines Merkmals (Scattering of a Feature)*: Summe der Anzahl von Paketen zur Implementierung einzelner Merkmale im Verhältnis zum Produkt der Anzahl an Paketen und Merkmalen [50]
FTANG [1]	*Verwicklung eines Pakets (Tangling of a Package)*: Summe der Anzahl von in einzelnen Paketen implementierten Merkmalen im Verhältnis zum Produkt der Anzahl an Paketen und Merkmalen [50]
AND [1]	*Durchschnittliche Verschachtelungstiefe von Annotationen (Average Nesting Depth of annotations)*: Durchschnittliche Verschachtelungstiefe aller Annotationen für das gesamte Projekt [32]

Tabelle 13: Merkmals-orientierte Komplexitätsmetriken

7.3 Anlage C - Kohäsionsmetriken

7.3.1 Anlage C.1 - Kohäsionsmetriken des High-Level-Designs

Metrik	Beschreibung
SCC [1]	*Ähnlichkeitsbasierte Klassenkohäsion (Similarity-Based Class Cohesion)*: Gewichtete Summe der Metriken MMAC, AAC, AMCOH und MMIC [7]
DCO [1]	*Kohäsionsgrad von Objekten (Degree of Cohesion Objects)*: Summe von *FANIN* jeder Klasse eines Pakets geteilt durch die gesamte Anzahl an Klassen des Pakets [51]
PLC	*Prozentualer Mangel an Kohäsion (Percentage Lack of Cohesion)*: 100 Prozent minus die durchschnittliche Kohäsion von Paket-Entitäten [2]
CAM, CAMC [1] [2]	*Kohäsion zwischen Methoden (Cohesion-Among-Methods)*: Berechnung der Verwandtschaft zwischen Methoden einer Klasse basierend auf deren Parameterlisten [23]
MMAC [1]	*Kohäsion zwischen Methoden durch Attribute (Method-Method through Attributes Cohesion)*: Durchschnittliche Kohäsion aller Methodenpaare einer Klasse [7]
CCRC [1]	*Relationale Kohäsion von Klassenkategorien (Class Category Relational Cohesion)*: Messung der Kohäsionsrate zwischen Klassen eines Paketes als Anzahl an Relationen zwischen Klassen im Verhältnis zur Anzahl aller Klassen des Pakets [52]
IUC [1]	*Kohäsion der Schnittstellenverwendung (Interface Usage Cohesion)*: Summe der relativen Anteile der durch Clienten verwendete Methoden einer Schnittstelle im Verhältnis zur gesamten Anzahl definierter Methoden und geteilt durch die Anzahl aller Clienten [53]
NHD [1]	*Normalisierte Hamming-Distanz (Normalized Hamming Distance)*: Berechnung der durchschnittlichen Übereinstimmung von Parametern zwischen jeden Methodenpaar. [7]

Tabelle 14: Kohäsionsmetriken des High-Level-Designs

7.3.2 Anlage C.2 - Kohäsionsmetriken des Low-Level-Designs

Metrik	Beschreibung
COH [1]	*Kohäsion (Cohesion)*: Berechnung der Kohäsion als Anteil der Anzahl unterschiedlicher, zugegriffener Attribute in den Methoden einer Klasse [54]
CCOH, COM [1]	*Klassenkohäsion (Class Cohesion)*: Summe der Anteile geteilter Attribute im Verhältnis zu unterschiedlichen Attributen zwischen allen Methodenpaaren im Verhältnis zu der Gesamtanzahl möglicher Methodenpaare [54]
SCOM [1]	*Sensitive Klassenkohäsion (Sensitive Class Cohesion)*: Identisch zu CCOH mit Verwendung eines anderen Ähnlichkeitsmaßes [54]
C3 [1]	*Konzeptionelle Kohäsion von Klassen (Conceptual Cohesion of Classes)*: Berechnung eines Grads der Ähnlichkeit zwischen Methoden einer Klasse hinsichtlich der Repräsentation einzelner semantischer Abbildungen [8]

Tabelle 14 – Fortsetzung der Kohäsionsmetriken des Low-Level-Designs

Metrik	**Beschreibung**
TCC [1]	*Enge Klassenkohäsion (Tight Class Cohesion)*: Relative Anzahl von Methodenpaaren einer Klasse mit direkter Verbindung zu einem Attribut durch dessen Verwendung innerhalb der Methodenkörper oder einer darin aufgerufenen Methode [9]
LCC [1]	*Lockere Klassenkohäsion (Loose Class Cohesion)*: Relative Anzahl von Methodenpaaren einer Klasse mit direkter oder indirekter Verbindung zu einem Attribut durch dessen Verwendung innerhalb der Methodenkörper oder einer darin aufgerufenen Methode bzw. über weitere Attribute und Methoden mit direkten Verbindungen [9]
LSCC [1]	*Kohäsion von Basisklassen auf niedriger Ebene (Low-level Similarity Base Class Cohesion)*: Summe der Anteile von geteilten Attribute und allen Klassenattributen aller Methodenpaare im Verhältnis zur Gesamtanzahl möglicher Methodenpaare [54]
PCCC [1]	*Klassenkohäsion durch Pfadkonnektivität (Path Connectivity Class Cohesion)*: Berechnung der Anzahl möglicher Pfade in einem Graphen zur Darstellung von Konnektivitätsmustern der Klassenmitglieder [54]
DCD [1]	*Grad direkter Kohäsion (Degree of Cohesion-Direct)*: Relative Anzahl von Methodenpaaren mit direkter Verbindung durch Attribute (siehe *TCC*) oder dem dirketen Aufruf der gleichen Methode [9]
DCI [1]	*Grad indirekter Kohäsion (Degree of Cohesion-Indirect)*: Relative Anzahl von Methodenpaaren mit direkter oder indirekter Verbindung durch Attribute (siehe *LCC*) oder dem dirketen Aufruf der gleichen Methode [9]
LCOM1 [1]	*Mangel an Kohäsion in Methoden 1 (Lack of Cohesion in Methods 1)*: Anzahl der Methodenpaare einer Klasse, die sich keine Instanz-Variablen teilen [54]
LCOM2 [1]	*Mangel an Kohäsion in Methoden 2 (Lack of Cohesion in Methods 2)*: Relative Anzahl an Methodenpaaren einer Klasse, welche sich Instanzvariablen teilen [54]
LCOM3 [1] [2]	*Mangel an Kohäsion in Methoden 3 (Lack of Cohesion in Methods 3)*: Berechnung der Anzahl an Methoden einer Klasse, welche sich Instanz-Variablen miteinander teilen [54]
LCOM4 [1]	*Mangel an Kohäsion in Methoden 4 (Lack of Cohesion in Methods 4)*: Berechnung der Anzahl an Methoden einer Klasse, welche sich Instanz-Variablen miteinander teilen oder eine Methode der Klasse aufrufen bzw. durch eine Methode der Klasse aufgerufen werden [54]
LCOM5 [1]	*Mangel an Kohäsion in Methoden 5 (Lack of Cohesion in Methods 5)*: Berechnung des Mangels an Kohäsion durch Betrachtung der Anzahl an Methoden einer mit Referenzen zu jedem Attribut [54]
TLCOM [1]	*Transitiver Mangel an Kohäsion in Methoden (Transitive LCOM)*: Relative Anzahl an Methodenpaaren einer Klasse, die sich Instanzvariablen direkt oder transitiv durch Methodenaufrufe teilen [55]

Tabelle 14 – Fortsetzung der Kohäsionsmetriken des Low-Level-Designs

Metrik	**Beschreibung**
CO [1]	*Konnektivität (Connectivity)*: Anzahl aller Methoden einer Klasse abzüglich der Anzahl der Methodenpaare, welche sich Instanz-Variablen teilen oder voneinander aufgerufen werden, im Verhältnis zur Anzahl dieser Methodenpaare [56]
AAC [1]	*Kohäsion zwischen Attributen (Attribute-Attribute Cohesion)*: Durchschnittliche Kohäsion aller Attributpaare einer Klasse [7]
AMCOH [1]	*Kohäsion zwischen Attributen und Methoden (Attribute-Method Cohesion)*: Durchschnittliche Anzahl von Attribut-Methoden-Interaktionen einer Klasse [7]
MMIC [1]	*Kohäsion zwischen Methoden mittels Methodenaufrufen (Method-Method Invocation Cohesion)*: Durchschnittliche Anzahl an Interaktionen als Methoden-Aufrufe durch andere Methoden einer Klasse [7]
CBMC [1]	*Kohäsion basierend auf Mitgliedskonnektivität (Cohesion Based on Member Connectivity)*: Produkt des Konnektivitätsfaktors und Strukturfaktors einer Klasse [57]
ICBMC [1]	*Verbesserte Kohäsion basierend auf Mitgliedskonnektivität (Improved Cohesion Based on Member Connectivity)*: Anpassung von Formeln der verwendeten Faktoren der *CBMC*-Metrik zur Bereitstellung verbesserter Metriken-Eigenschaften [57]
OLN [1]	*Durchschnittliche Stärke von Attributen(Average Strength of the Attributes)*: Berechnung durchschnittlicher Stärke von Attributen durch Berechnung der Stärke einzelner Attributen auf Grundlage der Stärke von Methoden, welche auf diese Attribute zugreifen [57]
COR [1]	*Kohäsion der Verantwortung (Cohesion of Responsibility)*: Reziprok der Anzahl an Mengen ähnlicher Methoden einer Klasse auf Basis verwendeter Attribute und Methoden [58]
ICH [1]	*Informationsbasierte Kohäsion (Information Based Cohesion)*: Anzahl an Aufrufen anderer Methoden der gleichen Klasse [13]
LMC	*Anzahl aufgerufener lokaler Methoden (Number of Local Methods Called)*: Anzahl der Aufrufe aller Methoden, die in derselben Klasse definierten worden [16]
MCC [1]	*Kohäsion von Methodengemeinschaften (Method Community Cohesion)*: Höchste Anzahl an Klassenmethoden der gleichen Gemeinschaft im Verhältnis zur Anzahl aller Methoden nach Anwendung eines Algorithmus zur Erkennung von Gemeinschaften [59]
MCEC [1]	*Entropier der Kohäsion von Methodengemeinschaften (Method Community Entropy Cohesion)*: Repräsentation der normalisierten Informations-Entropie auf Grundlage der Verteilung von Methoden einer Klasse in Gemeinschaften [59]

Tabelle 15: Kohäsionsmetriken des Low-Level-Designs

7.3.3 Anlage C.3 - Aspekt-orientierte Kohäsionsmetriken

Metrik	Beschreibung
LCOO [1]	*Mangel an Kohäsion in Operationen (Lack of Cohesion in Operations)*: Anzahl an Paaren von Operationen mit Verwendung verschiedener Felder von Aspekten oder Klassen minus die Anzahl an Operationspaaren mit Verwendung gemeinsamer Felder [47]
ACOH [1]	*Aspektkohäsionsmetrik (Aspect Cohesion Metric)*: Anzahl an Paaren von Modulen eines Aspekts mit Zusammenhang im Verhältnis zu allen Modulpaaren eines Aspekts [25]
PCOH [1]	*Kohäsion auf Paketebene (Package Level Cohesion)*: Durchschnitt der Anzahl an Relationen zwischen allen Klassen des Aspekts im Verhältnis zur gesamten Anzahl an Klassen des Aspekts [25]
UACOH [1]	*Einheitliche Aspektkohäsionsmetrik (Unified Aspect Cohesion Metric)*: Anzahl an Relationen zwischen den Bestandteilen einer Komponente im Verhältnis zur maximal möglichen Anzahl an Relationen [25]
LCBC [1]	*Mangel Concern-basierter Kohäsion (Lack of Concern-Based Cohesion)*: Anzahl der Concerns mit Adressierung durch die gegebene Komponente [60]

Tabelle 16: Aspekt-orientierte Kohäsionsmetriken

7.3.4 Anlage C.4 - Merkmals-orientierte Kohäsionsmetriken

Metrik	Beschreibung
IFD [1]	*Verhältnis interner Merkmals-Abhängigkeiten (Internal-Ratio Feature Dependency)*: Anzahl interner Abhängigkeiten im Verhältnis zu allen möglichen internen Abhängigkeiten [61]
EFD [1]	*Verhältnis externer Merkmals-Abhängigkeiten (External-Ratio Feature Dependency)*: Anzahl interner Abhängigkeiten im Verhältnis zu allen bestehenden Abhängigkeiten [61]

Tabelle 17: Merkmals-orientierte Kohäsionsmetriken

7.4 Anlage D - Kopplungsmetriken

7.4.1 Anlage D.1 - Allgemeine Kopplungsmetriken

Metrik	**Beschreibung**
COF, CF [1]	*Kopplungsfaktor (Coupling Factor)*: Relative Anzahl aktueller Kopplungen im Verhältnis zur maximal möglichen Anzahl an Kopplungen [13]
CBO [1] [2] [16]	*Kopplung zwischen Objekten (Coupling Between Objects)*: Anzahl aller mit einer Klasse durch Methodenaufrufe, Feldzugriffe, Vererbung, Argumenten, Rückgabewerten und Ausnahmebehandlungen gekoppelten Klassen [23]
DCC [1]	*Direkte Klassenkopplung (Direct Class Coupling)*: Anzahl unterschiedlicher Klassen mit direkter Relation zu einer Klasse inklusive Relationen zu Klassen hinsichtlich der Deklaration von Attributen und der Weitergabe von Nachrichten [62]
CCBC [1]	*Konzeptionelle Kopplung zwischen Klassen (Conceptual Coupling Between Classes)*: Berechnung von Kopplung zwischen Klassen basierend auf dem Grad der Beziehungen zwischen Elementen wie Kennzeichnern und Kommentaren [11]
NAS [1]	*Anzahl der Assoziationen pro Klasse (Number of Associations per Class)*: Anzahl aller Assoziationen einer Klasse mit anderen Klassen oder sich selbst [63]
NUCD [1]	*Anzahl der mittels Abhängigkeitsbeziehungen verwendeten Klassen (Number of Used Classes by Dependency Relation)*: Anzahl der durch eine Klasse verwendeten, unterschiedlichen Klassen mit Abhängigkeitsbeziehungen aller möglichen Arten (z. B.: Parameter, lokale Variablen, Rückgabetypen, etc.) [10]
NUCC [1]	*Anzahl der mittels Abhängigkeitsbeziehungen durch eine Klasse verwendeten Klassen (Number of Used Classes for a Class Through Dependency Relation)*: Gesamte Anzahl unterschiedlicher Klassen mit Verwendung einer Klasse mittels Abhängigkeitsbeziehungen [10]
MPC [1][16]	*Nachrichtendurchgangskopplung (Message Passage Coupling)*: Anzahl der zu anderen Klassen gesendeten Nachrichten [13]

Tabelle 18: Allgemeine Kopplungsmetriken

7.4.2 Anlage D.2 - Kopplungsmetriken basierend auf Datentypen und Feldzugriffen

Metrik	**Beschreibung**
CFA [1]	*Kopplung des Feldzugriffs (Coupling of Field Access)*: Anzahl an Klassen, Aspekten und Schnittstellen mit Deklaration eines Feldes mit Zugriff durch eine gegebene Klasse oder Aspekts [47]
DAC [1]	*Datenabstraktionskopplung (Data Abstraction Coupling)*: Anzahl definierter abstrakter Datentypen innerhalb einer Klasse [13]

Tabelle 18 – Fortsetzung der Kopplungsmetriken basierend auf Datentypen und Feldzugriffen

Metrik	Beschreibung
DAC2 [1]	*Datenabstraktionskopplung 2 (Data Abstraction Coupling 2)*: Anzahl unterschiedlicher Klassen unter Verwendung als Datentyp eines Attributs einer Klasse [57]
DAM [1] [2]	*Datenzugriffsmetrik (Data Access Metric)*: Anzahl an 'private' oder 'protected' deklarierten Attributen einer Klasse im Verhältnis zur gesamten Anzahl an deklarierten Feldern einer Klasse [23]
ACAIC [1]	*Importkopplung von Vorfahren durch Klassenattribute (Ancestor Class-Attribute Import Coupling)*: Anzahl an Klassen mit Attributen, welche eine Superklasse als Datentyp enthalten [56]
DCAEC [1]	*Exportkopplung von Nachfahren durch Klassenattribute (Descendant Class-Attribute Export Coupling)*: Anzahl an Klassen mit Attributen, welche in einer Unterklasse als Datentyp enthalten sind [56]
OCAEC [1]	*Exportkopplung durch Klassenattribute (Class-Attribute Export Coupling)*: Anzahl an Klassen mit Attributen, welche in einer Klasse ohne Vererbungsbeziehungen als Datentyp enthalten sind [56]
OCAIC [1]	*Importkopplung durch Klassenattribute (Class-Attribute Import Coupling)*: Anzahl an Klassen mit Attributen, welche eine Klasse ohne Vererbungsbeziehungen als Datentyp enthalten [56]
NOAV [1]	*Anzahl zugegriffener Variablen (Number Of Accessed Variables)*: Anzahl an in Klassen des Systems deklarierten Instanz-Variablen, lokalen und globalen Variablen sowie Parametern, auf welche innerhalb einer Methode entweder mittels Zugriffsmethoden oder direkt zugegriffen werden [64]

Tabelle 19: Kopplungsmetriken basierend auf Datentypen und Feldzugriffen

7.4.3 Anlage D.3 - Kopplungsmetriken basierend auf Methodenaufrufen

Metrik	Beschreibung
CA, FANIN, FIN [1] [2]	*Eingehende Kopplung (Afferent-Coupling)*: Anzahl aller anderen Klassen mit Methodenaufruf der gegebenen Klasse [23]
CE, FANOUT, FOUT [1] [2]	*Ausgehende Kopplung (Efferent-Coupling)*: Anzahl aller anderen Klassen mit Methodenaufruf durch die gegebene Klasse [23]
RMI [1]	*Instabilität (Instability)*: Berechnung mittels CA / (CA + CE)
DN, DMS, RMD [1]	*Normalisierte Distanz zur Hauptsequenz (Normalized Distance from Main Sequence)*: Normalisierung der Balance zwischen den Metriken der Abstraktion (A, RMA) und Instabilität (RMI) [65]
IC, CBI [1] [2]	*Kopplung basierend auf Vererbung (Coupling Based on Inheritance)*: Anzahl überschriebener oder nicht vererbter Methoden einer Klasse mit Kopplung zur vererbten Methoden [23]
RFC [1] [2] [16]	*Antworten für eine Klasse (Response for a class)*Anzahl der durch eine gegebene Klasse unterschiedliche, direkt aufgerufene Methoden [57]

Tabelle 19 – Fortsetzung der Kopplungsmetriken basierend auf Methodenaufrufen

Metrik	**Beschreibung**
CBM [1] [2]	*Kopplung zwischen Methoden (Coupling Between Methods)*: Anzahl an Eltern-Klassen mit Kopplung zur einer gegebenen Klasse durch den Aufruf einer Eltern-Methode [23]
CMC [1]	*Kopplung von Methodenaufrufen (Coupling of Method Call)*: Anzahl an Klassen, Aspekten und Schnittstellen mit Deklaration von Methoden mit potenziellen Aufruf durch eine gegebene Klasse oder Aspekts [47]
ACMIC [1]	*Importkopplung von Vorfahren durch Klassenmethoden (Ancestor Class-Method Import Coupling)*: Anzahl an Klassen mit Methoden, welche über eine Superklasse als Datentyp eines Parameters verfügen [56]
AMMIC [1]	*Importkopplung von Vorfahren durch Methodenaufrufe (Ancestor Method-Method Import Coupling)*: Anzahl an Klassen mit Methoden, welche eine Methode einer Superklasse aufrufen oder eine solche Methode als Parameter übergeben [56]
DCMEC [1]	*Exportkopplung von Nachfahren durch Klassenmethoden (Descendant Class-Method Export Coupling)*: Anzahl an Klassen mit Methoden, welche in einer Unterklasse als Datentyp eines Parameters verwendet werden [56]
DMMEC [1]	*Exportkopplung von Vorfharen durch Methodenaufrufe (Descendant Method-Method Export Coupling)*: Anzahl an Klassen mit Methoden, welche in einer Methode einer Unterklasse aufgerufen werden oder als Parameter übergeben werden [56]
OCMEC [1]	*Exportkopplung durch Klassenmethoden (Class-Method Export Coupling)*: Anzahl an Klassen mit Methoden, welche in einer Klasse ohne Vererbungsbeziehungen als Datentyp eines Parameters verwendet werden [56]
OCMIC [1]	*Importkopplung durch Klassenmethoden (Class-Method Import Coupling)*: Anzahl an Klassen mit Methoden, welche über eine Klasse ohne Vererbungsbeziehungen als Datentyp eines Parameters verfügen [56]
OMMEC [1]	*Exportkopplung durch Methodenaufrufe (Method-Method Export Coupling)*: Anzahl an Klassen mit Methoden, welche in einer Methode einer Klasse ohne Vererbungsbeziehungen aufgerufen werden oder als Parameter übergeben werden [56]
OMMIC [1]	*Importkopplung durch Methodenaufrufe (Method-Method Import Coupling)*: Anzahl an Klassen mit Methoden, welche eine Methode einer Klasse ohne Vererbungsbeziehungen aufrufen oder eine solche Methode als Parameter übergeben [56]
NOAM [1]	*Anzahl der Zugriffsmethoden (Number Of Accessor Methods)*: Anzahl aller Zugriffsmethoden (Getter- und Setter-Methoden) einer Klasse [64]
CINT [1]	*Kopplungsintensität (Coupling Intensity)*: Anzahl unterschiedlicher durch eine Methode aufgerufene Methoden [64]
CM [1]	*Methodenwechsel (Changing Methods)*: Anzahl unterschiedlicher Methoden mit Aufruf der gegebenen Methode [64]
CHC [1]	*Klassenwechsel (Changing Classes)*: Anzahl der Klassen von Methoden, welche die gegebene Methode aufrufen [64]

Tabelle 19 – Fortsetzung der Kopplungsmetriken basierend auf Methodenaufrufen

Metrik	**Beschreibung**
ICP [1]	*Informationsfluss basierend auf Kopplung (Information Flow Based Coupling)*: Anzahl an Methodenaufrufe innerhalb einer Klasse gewichtet anhand der Anzahl an Parametern aufgerufener Methoden [56]
EXT [16]	*Anzahl aufgerufener externer Methoden (Number of External Methods Called)*: Anzahl aller Aufrufe von in anderen Klassen definierten Methoden mit Ausnahme von Superklassen [16]
CWC [1]	*Kopplungsgewicht einer Klasse (Coupling Weight for a Class)*: Summe aus den kognitiven Gewichten aller Methodenaufrufe innerhalb einer Methode sowie den kognitiven Gewichten aufgerufener Methoden [45]
ACF [1]	*Durchschnittlicher Kopplungsfaktor (Average Coupling Factor)*: Summe der kognitiven Kopplungskomplexität (CWC) aller Methoden einer Klasse im Verhältnis zur Anzahl an Aufrufen von Methoden anderer Klassen [45]

Tabelle 20: Kopplungsmetriken basierend auf Methodenaufrufen

7.4.4 Anlage D.4 - Aspekt-orientierte Kopplungsmetriken

Metrik	**Beschreibung**
BAC [1]	*Basisaspektkopplung (Base-Aspect Coupling)*: Anzahl der für ein Aspekt durch Advices versteckten Joinpoints addiert mit der Anzahl an Änderungen der Modul-Hierarchie durch Aspekte mittels verschiedener Deklarationen [47]
CAE [1]	*Kopplung durch Advice-Ausführung (Coupling on Advice Execution)*: Anzahl an Aspekten mit Advices, welche möglicherweise durch die Ausführung von Operationen einer gegebenen Klasse oder Aspekts ausgelöst werden [47]
CD [1]	*Grad des Querschnitts einer Kopplung (Crosscutting Degree of a Concern)*: Anzahl der durch Pointcuts und Introductions betroffenen Komponenten für einen gegebenen Aspekt [26]
CDA [1]	*Grad des Querschnitts eines Aspekts (Crosscutting Degree of an Aspect)*: Anzahl an durch Pointcuts und Introduction eines gegebenen Aspekts betroffene Klassen und Aspekte [47]
DOS [1]	*Grad der Streuung (Degree of Scattering)*: Anzahl an Ziel-Elementen mit Adressierung des Quell-Elements im Verhältnis zur gesamten Anzahl an Elementen [60]
DOT [1]	*Verwicklungsgrad (Degree of Tangling)*: Anzahl an Quell-Elementen mit Adressierung durch ein bestimmtes Ziel-Element im Verhältnis zu allen Quell-Elementen [60]
FCD [1]	*Grad des Querschnitts eines Merkmals (Feature Crosscutting Degree)*: Anzahl der zur Realisierung eines Merkmals verwendeten Klassen [26]
CDC [1]	*Zerstreuung eines Concerns über Komponenten (Concern Diffusion over Components)*: Anzahl der Komponenten zur Implementierung eines Concerns [60]

Tabelle 20 – Fortsetzung aspekt-orientierter Kopplungsmetriken

Metrik	**Beschreibung**
CDO [1]	*Zerstreuung eines Concerns über Operationen (Concern Diffusion Over Operations)*: Anzahl an Methoden zur Implementierung eines Concerns [30]
CBC [1]	*Kopplung zwischen Komponenten (Coupling Between Components)*: Anzahl innerhalb einer Komponente (z. B.: Klassen, Aspekte) verwendeter Klassen zur Deklaration von Attributen sowie der Anzahl an deklarierten Komponenten in formalen Parametern, Rückgabetypen, lokalen Variablen und verwendeter Attribute und Methoden anderer Klassen oder Aspekte [66]
RFM [1]	*Antworten für ein Modul (Response for a Module)*: Anzahl an Methoden und Advices mit potenzieller Ausführung als Antwort auf eine durch eine gegebenen Klasse oder Aspekt erhaltene Nachricht [47]
CIM [1]	*Kopplung abgefangener Module (Coupling on Intercepted Modules)*: Anzahl an Modulen und Schnittstellen mit expliziter Erwähnung in Pointcuts eines gegebenen Aspekts [67]
CEA [1]	*Ausgehende Kopplung in Aspekten (Efferent Coupling in Aspects)*: Anzahl an Paketen (Aspekten), von denen ein gegebenes Paket (Aspekt) abhängt [25]
CCF [1]	*Kopplung aufgrund des Kontrollflusses (Coupling due to Control Flow)*: Berechnung der Kopplung eines Aspekts bezüglich der Veränderungen vom Kontrollfluss des Basissystems auf Grundlage der gesamten Anzahl an zu dem Aspekt hinzugefügten Advices sowie der gesamten Anzahl an ersetzenden und bedingt ersetzenden Advices innerhalb des Aspekts [68]
COS [1]	*Kopplung aufgrund des Objektzustands (Coupling due Object's State)*: Berechnung der Kopplung eines Aspekts bezüglich der Veränderung des Zustands eines Objekts des Basissystems auf Grundlage der gesamten Anzahl unabhängiger Advices, beobachtender Advices und gegensätzlichen Advices innerhalb des Aspekts [68]
CSS [1]	*Kopplung aufgrund der statischen Struktur (Coupling due to Static Structure)*: Berechnung der Kopplung eines Aspekts bezüglich der statischen Struktur des Basissystems auf Grundlage der gesamten Anzahl an Zwischentyp-Deklarationen von Zugriffsbeschränkungen, Injektionen, Schnittstellenmarkierungen, Rollenbereitstellungen und gen-spezifischen Veränderungen innerhalb des Aspekts [68]
CSC [1]	*Concern-sensitive Kopplung (Concern Sensitive Coupling)*: Anzahl an Komponenten mit Kopplung zu einer Klasse oder einem Aspekt, die den Concern realisiert [27]

Tabelle 21: Aspekt-orientierte Kopplungsmetriken

7.5 Anlage E - Vererbungsmetriken

7.5.1 Anlage E.1 - Vererbungsmetriken der Klassen-Ebene

Metrik	Beschreibung
SUBC, FANDOWN, NSUB [1]	*Anzahl an Unterklassen (Number of Subclasses)*: Anzahl der Unterklassen einer bestimmten Klasse [16]
SUPC, FANUP [1]	*Anzahl an Superklassen (Number of Superclasses)*: Anzahl der Superklassen einer bestimmten Klasse [16]
NOD [1]	*Anzahl an Nachfahren (Number of Descendents)*: Anzahl aller direkten und indirekten Unterklassen einer Klasse [56]
NSC [1] [2]	*Anzahl an Kindern (Number of Children)*: Anzahl aller direkten Unterklassen einer Klasse [23]
ANDC [1]	*Durchschnittliche Anzahl direkter Kinder (Average Number of Direct Child)*: Summe aller direkten Unterklassen im Verhältnis zur gesamten Anzahl an Klassen [12]
EANDC [1]	*Erweiterte durchschnittliche Anzahl direkter Kinder (Extended Average Number of Direct Child)*: Durchschnittliche Anzahl der mit direkten Unterklassen geteilten Attribute und Methoden [12]
ANIC [1]	*Durchschnittliche Anzahl indirekter Kinder (Average Number of Indirect Child)*: Summe aller indirekten Unterklassen im Verhältnis zur gesamten Anzahl an Klassen [12]
EANIC [1]	*Erweiterte durchschnittliche Anzahl indirekter Kinder (Extended Average Number of Indirect Child)*: Durchschnittliche Anzahl der mit indirekten Unterklassen geteilten Attribute und Methoden [12]
NOA [1]	*Anzahl an Vorfahren (Number of Ancestors)*: Anzahl aller direkten und indirekten Superklassen einer Klasse [56]
ANA, ACA [1]	*Durchschnittliche Anzahl an Vorfahren (Average Number of Ancestors)*: Durchschnittliche Anzahl an Klassen, welche Informationen direkt oder indirekt von Superklassen erben [62]
NP [1]	*Anzahl an Eltern (Number of Parents)*: Anzahl aller direkten Superklassen einer Klasse [56]

Tabelle 22: Vererbungsmetriken der Klassen-Ebene

7.5.2 Anlage E.2 - Vererbungsmetriken der Attribut- und Methoden-Ebene

Metrik	Beschreibung
MIF [1]	*Methodenvererbungsfaktor (Method Inheritance Factor)*: Summe aller in Klassen vererbten Methoden im Verhältnis zur Summe aller in Klassen vererbten und deklarierten Methoden [13]
MFA [1] [2]	*Maß funktionaler Abstraktion (Measure-of-Functional-Abstraction)*: Anzahl geerbter Methoden einer Klasse im Verhältnis zur gesamten Anzahl durch eine Klasse zugreifbare Methoden [23]

Tabelle 22 – Fortsetzung der Vererbungsmetriken der Attribut- und Methoden-Ebene

Metrik	**Beschreibung**
CMIF [1]	*Methodenvererbungsfaktor von Klassen (Class Method Inheritance Factor)*: Summe der neudefinierten Methoden aller Klassen im Verhältnis zur Summe aller verfügbaren Methoden aller Klassen [1]
NMI, NOMI [1] [2]	*Anzahl geerbter Methoden (Number of Methods Inherited)*: Anzahl an Methoden einer Unterklasse, welche von ihren Basisklassen vererbt und nicht überschrieben wurden [56]
NMO, NORM, NOVM, NOOC [1]	*Anzahl überschriebener Methoden (Number of Overriden Methods)*: Anzahl an Methoden einer Unterklasse, welche Methoden ihrer Basisklassen überschreiben [56]
NMA, NNM [1]	*Anzahl hinzugefügter Operationen (Number of Operations Added)*: Anzahl an Methoden einer Klasse, welche weder vererbt noch überschrieben wurden [56]
NOP, CPM [1]	*Anzahl polymorpher Methoden (Number of Polymorphic Methods)*: Anzahl an Methoden mit polymorphen Verhalten [62]
PF [1]	*Polymorphiefaktor (Polymorphism Factor)*: Anzahl aktuell überschriebener Methoden im Verhältnis zur maximal möglichen Anzahl überschriebener Methoden [13]
PPF [1]	*Parametrischer Polymorphiefaktor (Parametric Polymorphism Factor)*: Anzahl aller generischen Klassen im Verhältnis zur gesamten Anzahl an Klassen [69]
AIF [1]	*Attributvererbungsfaktor (Attribute Inheritance Factor)*: Summe aller in Klassen vererbten Attribute im Verhältnis zur Summe aller in Klassen vererbten und deklarierten Attribute [13]
NOAI, NIA	*Anzahl geerbter Attribute (Number of Attributes Inherited)*: Anzahl an Attribute einer Unterklasse, welche von ihren Basisklassen vererbt wurden [1] [2]
NAA [1]	*Anzahl hinzugefügter Attribute (Number of Attributes Added)*: Anzahl an Attributen einer Klasse, welche nicht vererbt wurden [56]

Tabelle 23: Vererbungsmetriken der Attribut- und Methoden-Ebene

7.5.3 Anlage E.3 - Metriken der Vererbungshierarchie

Metrik	**Beschreibung**
A, RMA [1]	*Abstraktheit (Abstractness)*: Relative Anzahl abstrakter Datentypen (abstrakte Klassen und Schnittstellen) hinsichtlich der gesamten Anzahl an Datentypen im System [70]
DIT [1] [2]	*Tiefe des Vererbungsbaums (Depth of Inheritance Tree)*: Die maximale Länge eines Pfades innerhalb eines Vererbungsbaumes beginnend bei einer Klasse bis hin zur Wurzel dieser Struktur [16]

Tabelle 23 – Fortsetzung der Metriken der Vererbungshierarchie

Metrik	Beschreibung
DIC [1]	*Vererbungsgrad einer Klasse (Degree of Inheritance of Class)*: Anzahl der in einer Klasse direkt geerbten Attribute in Abhängigkeit ihrer maximalen Vererbungstiefe [71]
AID [1]	*Durchschnittliche Vererbungstiefe einer Klasse (Average Inheritance Depth of a Class)*: Wert ist für Klassen ohne Vorfahren Null und für alle anderen Klassen der Durchschnitt der AID-Werte aller Superklassen addiert mit Eins [56]
CLD [1]	*Blattiefe von Klassen (Class to Leaf Depth)*: Länge eines Pfades von einer Superklasse aus bis hin zur entferntesten Unterklasse [72]
IIF [1]	*Interner Vererbungsfaktor (Internal Inheritance Factor)*: Relative Anzahl aller Klassen mit Vererbungen bezüglich Klassen des gleichen Systems im Verhältnis zu allen Klassen mit Vererbungen [13]
CIF [1]	*Vererbungsfaktor einer Klasse (Class Inheritance Factor)*: Anzahl aller Unterklassen im Verhältnis zur gesamten Anzahl verfügbarer Klassen [73]
DBRM [1]	*Verhältnismetrik zwischen abgeleiteten Klassen und Basisklassen (Derive Base Ratio Metric)*: Anzahl aller abgeleiteten Klassen im Verhältnis zur Anzahl aller Basisklassen [12]
EDBRM [1]	*Erweiterte Verhältnismetrik zwischen abgeleiteten Klassen und Basisklassen (Extended Derived Base Ratio Metrics)*: Durchschnittliches Verhältnis zwischen der Anzahl aller geteilten Attribute und Methoden abgeleiteter Klassen und Basisklassen [12]
SIX, SI [1]	*Spezialisierungsindex (Specialization index)*: *NMO * DIT / (NMO + NMA + NMI)* [72]
SPR [1]	*Spezialisierungsverhältnis (Specialization Ratio)*: Anzahl an Unterklassen im Verhältnis zur Anzahl an Superklassen [72]
RR [1]	*Wiederverwendungsverhältnis (Reuse Ratio)*: Anzahl an Superklassen im Verhältnis zur gesamten Anzahl an Klassen [72]
NOH, NH [1]	*Anzahl an Hierarchien (Number of Hierarchies)*: Gesamte Anzahl aller nicht erbenden Klassen mit Unterklassen innerhalb einer Design-Spezifikation [17]
HIER	*Anzahl an Methodenaufrufen der Klassenhierarchie (Number of Class Hierarchy Methods Called)*: Anzahl aller Aufrufe von in Superklassen definierten Methoden [16]

Tabelle 24: Metriken der Vererbungshierarchie

7.5.4 Anlage E.4 - Aspekt-orientierte Vererbungsmetriken

Metrik	Beschreibung
APIF [1]	*Vererbungsfaktor von Aspekten (Aspect Inheritance Factor)*: Anzahl konkreter Aspekte im Verhältnis zur Anzahl aller verfügbaren Aspekte [73]
ADIF [1]	*Vererbungsfaktor von Advices (Advice Inheritance Factor)*: Anzahl neudefinierter Advices aller Aspekte im Verhältnis zur Anzahl verfügbarer Advices aller Aspekten [73]

Tabelle 24 – Fortsetzung aspekt-orientierter Vererbungsmetriken

Metrik	**Beschreibung**
PIF [1]	*Vererbungsfaktor von Pointcuts (Pointcut Inheritance Factor)*: Anzahl neudefinierter Pointcuts aller Aspekte im Verhältnis zur Anzahl verfügbarer Pointcuts aller Aspekten [73]
ATTIF [1]	*Vererbungsfaktor von Attributen (Attribute Inheritance Factor)*: Anzahl neudefinierter Attribute aller Aspekte im Verhältnis zur Anzahl verfügbarer Attribute aller Aspekte [73]
NOCOA [1]	*Anzahl an Kindern eines Aspekts (Number of Children of the Aspect)*: Anzahl direkter Unteraspekte eines gegebenen Moduls [47]

Tabelle 25: Aspekt-orientierte Vererbungsmetriken

7.6 Anlage F - Sicherheitsmetriken

7.6.1 Anlage F.1 - Zugriffsmetriken

Metrik	Beschreibung
MOD	*Anzahl an Modifikatoren (Number of Modifiers)*: Anzahl verwendeter Zugriffsmodifikatoren (z. B.: public, protected, etc.) zur Deklaration einer Klasse [16]
CIS [1]	*Größe der Klassenschnittstelle (Class Interface Size)*: Anzahl öffentlicher (public) Methoden einer Klasse innerhalb einer Design-Spezifikation [17]
ATFD [1]	*Zugriff auf fremde Daten (Access To Foreign Data)*: Anzahl an Attributen aus Klassen des Systems ohne Relationen mit direkten Zugriff oder durch Aufruf einer Zugriffsmethode [64]
NOA, NF [1] [2]	*Anzahl an Attributen (Number of Attributes)*: Anzahl aller Attribute einer Klasse
LVAR [1]	*Anzahl lokaler Variablen (Local Variables)*: Anzahl lokal deklarierter Variablen innerhalb einer Methode [18]
NIV, INST [1] [2]	*Anzahl an Instanz-Variablen (Number of Instance Variables)*: Anzahl aller Instanz-Variablen einer Klasse [16]
NOPA, PUBA, NOAP, NPV [1] [2]	*Anzahl öffentlicher Attribute (Number of Public Attributes)*: Anzahl aller innerhalb einer Klasse als 'public' deklarierte Attribute [64]
CDCIVPT [2]	*Anzahl deklarierter geschützer Instanz-Variablen (CountDeclInstanceVariableProtected)*: Anzahl der als 'protected' deklarierten Instanz-Variablen [2]
NOPRA [1] [2]	*Anzahl privater Attribute (Number of Private Attributes)*: Anzahl aller innerhalb einer Klasse als 'private' deklarierte Attribute
NSF [1]	*Anzahl statischer Attribute (Number of Static Attributes)*: Anzahl aller innerhalb einer Klasse als 'private' deklarierte Attribute
NOM, NM, MPC, NCM, TNM [1] [2]	*Anzahl an Klassen-Methoden (Number of Class Methods)*: Anzahl an in einer Klasse implementiere Methoden mit ausschließlichen Zugriff auf zu der Klasse selbst zugehörigen Daten [5]
NIM [1] [2]	*Anzahl an Instanz-Methoden (Number of Instance Methods)*: Anzahl an in einer Klasse implementierten Methoden mit Zugriff auf lokale Daten durch eine Instanz als Objekt der Klasse [5]
CDCM [2]	*Anzahl deklarierter Methoden (CountDeclMethod)*: Anzahl lokal deklarierter Methoden [2]
NOPM, NPM, PM [1] [2]	*Anzahl öffentlicher Methoden (Number of Public Methods)*: Anzahl aller innerhalb einer Klasse als 'public' deklarierten Methoden [23]
CDCMPT [2]	*Anzahl deklarierter geschützter Methoden (CountDeclMethodProtected)*: Anzahl der als 'protected' deklarierten, lokalen Methoden [2]
NOPRM [1] [2]	*Anzahl privater Methoden (Number of Private Methods)*: Anzahl aller innerhalb einer Klasse als 'public' deklarierten Methoden
NSM [1]	*Anzahl statischer Methoden (Number of Static Methods)*: Anzahl aller innerhalb einer Klasse als 'static' deklarierten Methoden

Tabelle 25 – Fortsetzung der Zugriffsmetriken

Metrik	Beschreibung
CDCMC [2]	*Anzahl deklarierter konstanter Methoden (CountDeclMethodConst)*: Anzahl der als konstant deklarierten, lokalen Methoden [2]
CDCMF [2]	*Anzahl deklarierter Freundesmethoden (CountDeclMethodFriend)*: Anzahl der lokal deklarierten Friend-Methoden [2]
IP [1]	*Datentransfer (Messaging)*: Anzahl öffentlicher Klassen eines Aspekts als Schnittstellengröße des Paktes [25]

Tabelle 26: Zugriffsmetriken

7.6.2 Anlage F.2 - Kapselungsmetriken

Metrik	Beschreibung
MHF [1]	*Methodenverbergungsfaktor (Method Hiding Factor)*: Zählung der durchschnittlichen Menge an versteckten Methoden in allen Klassen des Systems [13]
AHF	*Attributverbergungsfaktor (Attribute Hiding Factor)*: Zählung der durchschnittlichen Menge an versteckten Attributen in allen Klassen des Systems [13]
AHEF [1]	*Effiktivitätsfaktor zur Verbergung von Attributen (Attribute Hiding Effectiveness Factor)*: Anzahl an Klassen mit Zugriff auf die Attribute einer Klasse im Verhältnis zur gesamten Anzahl an Klassen, für die ein Zugriff auf die Attribute einer Klasse möglich ist [69]
OHEF [1]	*Effiktivitätsfaktor zur Verbergung von Operationen (Operation Hiding Effectiveness Factor)*: Anzahl an Klassen mit Aufruf von Methoden einer Klasse im Verhältnis zur gesamten Anzahl an Klassen, für die der Aufruf von Methoden einer Klasse möglich ist [69]
MOA [1]	*Aggregationsmaß(Measure of Aggregation)*: Anzahl an Deklarationen von Daten mit benutzerdefinierten Klassen als Datentypen [2]

Tabelle 27: Kapselungsmetriken

7.6.3 Anlage F.3 - Metriken der Ausnahmebehandlung

Metrik	Beschreibung
NOTB, NOT [1]	*Anzahl an Try-Blöcken (Number of Try Blocks)*: Anzahl aller Try-Blöcke eines Systems oder einer Komponente [74]
NOCB [1]	*Anzahl an Catch-Blöcken (Number of Catch Blocks)*: Anzahl aller Catch-Blöcke eines Systems oder einer Komponente [74]
NOFB [1]	*Anzahl an Finally-Blöcken (Number of Finally Blocks)*: Anzahl aller Finally-Blöcke eines Systems oder einer Komponente [74]
LOCEH [1]	*Anzahl der Code-Zeilen mit Bezug zur Ausnahmebehandlungs-Code (Number of LOC Pertaining to Exception-Handling Code)*: Anzahl aller für Ausnahmebehandlungen implementierten Code-Zeilen [1]

Tabelle 27 – Fortsetzung der Metriken der Ausnahmebehandlung

Metrik	**Beschreibung**
CAAEC [1]	*Anzahl durchschnittlicher Ausnahmen von Catch-Bereichen einer Klasse (Catch Area Average Exception in a Class)*: Anzahl aller Catch-Blöcke einer Klasse im Verhältnis zu ihrer Anzahl an Methoden [75]

Tabelle 28: Metriken der Ausnahmebehandlung

7.7 Anlage G - Netzwerkmetriken

7.7.1 Anlage G.1 - Metriken des Ego-Netzwerks

Metrik	**Beschreibung**
SIZE [1]	*Anzahl an Knoten (Number of Nodes)*: Anzahl der Knoten im Ego-Netzwerk eines bestimmten Knotens [16]
TIES [1]	*Anzahl an Kanten (Number of Edges)*: Anzahl an Kanten im Ego-Netzwerk eines bestimmten Knotens [16]
PAIRS [1]	*Anzahl an geordneten Paaren (Number of Ordered Pairs)*: Anzahl geordneter Paare als maximale Anzahl gerichteter Kanten (z. B.: $SIZE * (SIZE - 1)$) [16]
DN [1]	*Dichte (Density)*: *TIES* im Verhältnis zur maximal möglichen Anzahl an Kanten des Ego-Netzwerks [16]
WCP [1]	*Anzahl schwacher Komponenten (Weak Components)*: Anzahl disjunkter Mengen von Knoten im Ego-Netzwerk eines bestimmten Knotens ohne Betrachtung des Ego-Knotens und seiner Kanten [16]
NWCP	*Normalisierte Anzahl schwacher Komponenten (Normalized Weak Components)*: *WCP* im Verhältnis zur Anzahl an Knoten des Ego-Netzwerks (*SIZE*) [16]
TSR [1]	*Reichweite in zwei Schritten (Two Step Reach)*: Anzahl an Knoten innerhalb eines zweistufigen Abstands [16]
REFF [1]	*Reichweiteneffizienz (Reach-Efficiency)*: Anzahl an Knoten innerhalb eines zweistufigen Abstands (*TSR*) geteilt durch *SIZE* [76]
BRKG [1]	*Vermittlung (Brokerage)*: Anzahl an Knoten-Paaren, die im Ego-Netzwerk eines bestimmten Knotens nicht verbunden sind [16]
NBRKG	*Normalisierte Vermittlung (nBrokerage)*: *BRKG* im Verhältnis zur maximal möglichen Anzahl an Knoten-Paaren des Ego-Netzwerks [16]
EB [1]	*Ego dazwischen (EgoBetween)*: Anzahl der kürzesten Pfade von Nachbarn des Ego-Knotens mit Verlauf durch das Ego-Netzwerk als Prozentangabe [16]
NEB	*Normalisiertes Ego dazwischen (nEgoBetween)*: Normalisierung von *EB* anhand von *SIZE* [16]

Tabelle 29: Metriken des Ego-Netzwerks

7.7.2 Anlage G.2 - Globale Netzwerkmetriken

Metrik	**Beschreibung**
N [1]	*Anzahl an Knoten (Number of Nodes)*: Anzahl aller Knoten im globalen Netzwerk [77]
L [1]	*Anzahl an Kanten (Number of Edges)*: Anzahl aller Kanten im globalen Netzwerk [77]

Tabelle 29 – Fortsetzung globaler Netzwerkmetriken

Metrik	Beschreibung
ASP [1]	*Durchschnittlicher kürzester Pfad (Average Shortest Path)*: Durchschnittliche Länge aller kürzesten Pfade beliebiger Knoten im Verhältnis zu *PAIRS* [42]
CLC [1]	*Clustering-Koeffizient (Clustering Coefficient)*: Berechnung der Ähnlichkeit einzelner Ego-Netzwerke hinsichtlich einer Clique und dem Durchschnitt dieser Werte aller Ego-Netzwerke als Clustering Coefficient des globalen Netzwerks [77]
MDM [1]	*Metrik mehrerer Abhängigkeiten (Multiple Dependency Metric)*: *IDC* multipliziert mit *REACH* im Verhältnis zu dem Produkt der maximalen Werte von IDC und REACH des globalen Netzwerks [78]

Tabelle 30: Globale Netzwerkmetriken

7.7.3 Anlage G.3 - Metriken struktureller Löcher

Metrik	Beschreibung
EFFSZ [1]	*Effektive Größe (Effective Size)*: Anzahl an Knoten im Ego-Netzwerk eines bestimmten Knotens abzüglich der durchschnittlichen Anzahl an Kanten der Knoten des Netzwerks [16]
EFF [16]	*Effizienz (Efficiency)*: *EFFSZ* im Verhältnis zur Anzahl an Knoten des globalen Netzwerks
CSTR [1]	*Zwang (Constraint)*: Berechnung der Stärke von Einschränkungen eines Knotens durch seine Nachbarn hinsichtlich der Erreichbarkeit weiterer Knoten [16]
H [1]	*Hierarchie (Hierarchy)*: Berechnung der Konzentration von Beschränkungen eines Knotens durch seine Nachbarn im globalen Netzwerk [16]

Tabelle 31: Metriken struktureller Löcher

7.7.4 Anlage G.4 - Zentralitätsmetriken

Metrik	Beschreibung
D [1]	*Grad (Degree)*: Anzahl der Kanten einer Entität [16]
ND	*Normalisierter Grad (nDegree)*: Anzahl der Kanten einer Entität im Verhältnis zur gesamten Anzahl an Entitäten [16]
IDC [1]	*Ausgehender Zentralitätsgrad (In Degree Centrality)*: Anzahl aller von einer Entität ausgehenden, gerichteten Kanten [42]
ODC [1]	*Eingehender Zentralitätsgrad (Out Degree Centrality)*: Anzahl aller zu einer Entität eingehenden, gerichteten Kanten [42]
C [1]	*Nähe (Closeness)*: Summe der Länge aller kürzesten Pfade beginnend bei einer Entität zu jeder anderen Entität des Netzwerks [16]
CC [1]	*Zentralität der Nähe (Closeness Centrality)*: Mittelwert der kürzesten Pfade beginnend bei einer Entität zu jeder anderen Entität des Netzwerks [42]

Tabelle 31 – Fortsetzung der Zentralitätsmetriken

Metrik	**Beschreibung**
I [1]	*Information (Information)*: Harmonisches Mittel der Länge von Pfaden endend bei einer gegebenen Entität [16]
INFCY [1]	*Informationszentralität (Information Centrality)*: Harmonisches Mittel der Länge von allen Pfaden beginnend bei einer Entität zu jeder anderen Entität des Netzwerks [76]
REACH [1]	*Erreichbarkeit (Reachability)*: Anzahl der von einer gegebenen Entität erreichbaren Entitäten [16]
DWRH [1]	*Erreichbarkeit als Bruch (dwReach)* - Anzahl der von einer gegebenen Entität erreichbaren Entitäten gewichtet anhand der dafür benötigten Schritte als Brüche (z. B.: 1/1, 1/2, ...) [76]
B [1]	*Dazwischen (Betweenness)*: Anzahl der Auftreten einer gegebenen Entität auf den kürzesten Pfaden beliebiger anderer Entitäten [16]
NB	*Noramlisiertes Dazwischen (nBetweenness)*: Normalisierung von B anhand von $PAIRS$ [16]
BCD [1]	*Streuung der Zwischenzentralität (Betweenness Centrality Dispersion)*: Summe der Negationen von NB $(1 - NB)$ aller Knoten [79]
BCN [1]	*Zwischenzentralität (Betweenness Centralization)*: Normalisierung von BCD anhand des maximal möglichen BCD-Wertes für das gegebene Netzwerk [79]

Tabelle 32: Zentralitätsmetriken

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Seitens des Verfassers bestehen keine Einwände die vorliegende Hausarbeit für die öffentliche Benutzung im Universitätsarchiv zur Verfügung zu stellen.

Jena, den 15. Dezember 2022