Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Qualitätsanalyse von Studienprojekten anhand von Quellcode-Repositories

Benedikt Kersjes

Studiengang:	Softwaretechnik
Prüfer:	Prof. Stefan Wagner
Betreuer:	Dr. Ivan Bogicevic, Dr. Falko Kötter
Beginn am:	10. Mai 2017
Beendet am:	10. November 2017
CR-Nummer:	D.2.9

Kurzfassung

An vielen Hochschulen werden studentische Softwareentwicklungsprojekte als Teil der akademischen Ausbildung in Studiengängen wie Informatik oder Softwaretechnik durchgeführt. Auch an der Universität Stuttgart und insbesondere am Fraunhofer IAO werden im Bachelorstudiengang Softwaretechnik Studienprojekte durchgeführt. Da das Institut die Studienprojekte als Grundlage ihrer weiteren Forschungstätigkeit verwendet, besteht ein Interesse, dass in den Projekten qualitativ hochwertige Software entsteht.
In dieser Arbeit wurde die Qualität von sechs Studienprojekten anhand ihres Quellcodes analysiert, die in den letzten Jahren am Fraunhofer IAO durchgeführt wurden. Darüber hinaus wurde eine Analyse der Repositories durchgeführt und eine Befragung der Betreuer vorgenommen, um mögliche Einflussfaktoren auf die Qualität der Projekte zu ermitteln. Hierzu wurde ein Werkzeug entwickelt, das die Analyse automatisiert durchführt und verwendet werden kann, um zukünftige Projekte am Institut nach der gleichen Methodik auszuwerten.
Die Analysen ergaben, dass die gleichmäßige Verteilung der Arbeit über das Projekt, die Teilnehmerzahl, die Qualifikation der Betreuer und die Verwendung von Drittanbieter-Bibliotheken einen hohen Einfluss auf die Qualität der untersuchten Projekte hatten. Aus diesen Erkenntnissen wurden Handlungsempfehlungen für die Betreuung und Durchführung zukünftiger Studienprojekte am Fraunhofer IAO abgeleitet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1. Motivation

Studentische Softwareentwicklungsprojekte gehören international an vielen Hochschulen in Studiengängen wie Informatik oder Softwaretechnik zum festen Bestandteil der akademischen Ausbildung. Die Ziele dieser Projekte sind vielfältig: einige vermitteln allgemeine, praktische Techniken zur systematischen Arbeit im Team [BHSV04; BKA15], andere haben konkretere Lernziele wie den Umgang mit verteilten Projekten [BDKT00], Reverse Engineering [CLC09] oder die Wartung bereits existierender Systeme, um die Studierenden besser auf die Arbeitsweise in der Industrie vorzubereiten [DHZS99; GSM13].

Auch an der Universität Stuttgart gehört im Bachelor-Studiengang Softwaretechnik ein sogenanntes Studienprojekt zum Lehrplan. Ziel dieses Projektes ist ebenfalls die Vorbereitung auf berufstypische Arbeitsweisen und die Vermittlung von Kompetenzen zur verantwortlichen Teamarbeit [Fac]. Die Studienprojekte werden von verschiedenen Instituten der Universität betreut. Auch das Fraunhofer-Institut IAO, das diese Arbeit in Zusammenarbeit mit der Abteilung SE der Universität betreut, führt regelmäßig Studienprojekte durch.

In einem Studienprojekt entwickelt ein Team von 8 bis 15 Studierenden über zwei Semester eine Software. Studierende, die ein Studienprojekt beginnen, sind meist mindestens im vierten Semester. Daher sind die theoretischen Grundlagen und auch erste praktische Erfahrungen aus Projekten in niedrigeren Semestern bei den meisten Teilnehmern vorhanden. Das Projekt ist auf eine Arbeitszeit von etwa zehn Stunden pro Woche und Teilnehmer ausgelegt.
Am Fraunhofer IAO wird als Vorgehensmodell durchgängig eine abgewandelte Form von Scrum eingesetzt. Vor und nach jedem Sprint werden verpflichtende Termine ausgemacht, an denen die Ergebnisse des letzten Sprints präsentiert werden und die Aufgaben für den nächsten Sprint besprochen werden. Eine Vorgabe für die meisten Projekte ist außerdem, dass sich die Studierenden mindestens einen Tag pro Woche in einem reservierten Raum am Fraunhofer IAO treffen, um gemeinsam am Projekt zu arbeiten.

Neben den Studierenden, die sich von einem erfolgreichen Projekt einen hohen Wissenszuwachs und eine gute Note erhoffen, hat auch das Fraunhofer IAO ein Interesse an qualitativ hochwertigen Projekten. Das Institut führt diese Projekte häufig im Rahmen seiner Forschungstätigkeit durch und möchte die gewonnenen Erkenntnisse für die weitere Forschung nutzen. Beide Seiten profitieren also von einer hohen Qualität der entstehenden Software.

Da die Qualität der entstehenden Software, über die Art und den Umfang der Betreuung, die Themenauswahl und die Teamzusammensetzung maßgeblich vom Institut beeinflusst wird, entsteht das Bedürfnis, diese Einflussfaktoren zu analysieren und zu verbessern, um damit beste Voraussetzungen für erfolgreiche Studienprojekte zu schaffen.

1.2. Aufgabenstellung

Aus dem Ziel heraus, die Betreuung der am Fraunhofer IAO durchgeführten Studienprojekte zu evaluieren, entstand die Idee für diese Arbeit und leitet sich deren Aufgabenstellung ab. Zunächst soll die Arbeit einen Überblick über die Qualität der Studienprojekte geben, die in den vergangenen Jahren am Fraunhofer IAO durchgeführt wurden. Dazu wird eine Analyse des Quellcodes der betreffenden Projekte durchgeführt. Diese Analyse soll möglichst automatisiert, mithilfe eines Werkzeugs, das hierzu entwickelt wird, durchgeführt werden und wenige manuelle Schritte erfordern. Dadurch ist gewährleistet, dass auch in den nächsten Jahren neue Projekte nach der gleichen Methodik analysiert werden können und eine Entwicklung in der Qualität der Projekte erkennbar wird.

Nach Conway's Law [Con68] ist die Struktur eines Software-Systems durch die Kommunikationsstrukturen der Entwicklungsorganisation vorbestimmt. Dies legt nahe, dass auch der Entstehungsprozess der Studienprojekte Einfluss auf die Struktur und die Qualität des Quellcodes hat. Daher wird in dieser Arbeit auch der Einfluss des Entstehungsprozesses auf die Qualität der Projekte untersucht. Hierzu werden die Quellcode-Repositories der Projekte analysiert, da diese durch den Versionsverlauf Rückschlüsse auf die Entstehung der Projekte zulassen.

Eine dritte Untersuchung soll Eigenschaften der Betreuung der Projekte erheben. Dazu wird eine Umfrage unter den Betreuern der untersuchten Projekte durchgeführt, bei der zum einen die Rahmenbedingungen der Projekte erhoben werden als auch eine Einschätzung der Qualität und der Betreuung erbeten wird.

Diese Arbeit basiert auf der Annahme, dass sowohl der Entstehungsprozess als auch der Betreuungsprozess einen Einfluss auf die Qualität der entstehenden Software haben. Um aus den Erkenntnissen dieser Arbeit zu lernen, werden aus den ermittelten Einflussfaktoren Handlungsempfehlungen abgeleitet, die die Betreuer der Studienprojekte am Fraunhofer IAO bei der Durchführung zukünftiger Studienprojekte unterstützen sollen.

Bei der vorliegenden Arbeit handelt es sich um eine explorative Untersuchung, da die untersuchten Projekte sehr unterschiedlich sind und insgesamt nur wenige Projekte untersucht wurden. Eine Vergleichbarkeit und Allgemeingültigkeit ist daher nur bedingt gewährleistet. Die Erkenntnisse liefern aber Anhaltspunkte, um mit kontinuierlicher Anwendung des Werkzeugs die Prozesse nachhaltig zu verbessern.

1.3. Gliederung

Die vorliegende Arbeit ist in folgender Weise gegliedert:

Kapitel 1 - Einleitung Dieses Kapitel legt die Motivation und Aufgabenstellung dieser Arbeit dar.

Kapitel 2 - Stand der Wissenschaft und Technik Der wissenschaftliche Kontext der Arbeit wird untersucht, um die Arbeit einordnen zu können.

Kapitel 3 - Methodik Die Methodik zur Analyse der Projekte wird erarbeitet. Es werden die drei Analysen Qualitätsanalyse, Repository-Analyse und Betreuungsanalyse vorgestellt.

Kapitel 4 - Implementierung Die Implementierung der Qualitäts- und der Repository-Analyse wird vorgestellt. Die Implementierung wurde zur Auswertung der Projekte verwendet.

Kapitel 5 - Analyse und Evaluation Die Ergebnisse der drei Analysen werden präsentiert und es werden Empfehlungen zur Verbesserung der Betreuung und Durchführung zukünftiger Projekte abgeleitet.

Kapitel 6 - Zusammenfassung und Ausblick Dieses Kapitel fasst die Ergebnisse zusammen und liefert Anregungen für weitere Untersuchungen.

2. Stand der Wissenschaft und Technik

Dieses Kapitel beschäftigt sich mit verwandten wissenschaftlichen Arbeiten und allgemeinen Qualitätsmodellen, Metriken und Werkzeugen, um diese Arbeit inhaltlich einzuordnen und eine Grundlage zu bieten, um eine eigene Methodik zur Lösung der Problemstellung abzuleiten. Die Auswahl der Arbeiten wurde mithilfe einer ausführlichen Literaturrecherche zu den Themen dieser Arbeit getroffen. Der Fokus wurde auf verbreitete und populäre Beiträge gelegt, die eine hohe Relevanz für diese Arbeit haben.

2.1. Verwandte Arbeiten

Im Jahr 2006 führte Hampp [Ham06] bereits eine Analyse von Studienprojekten an der Universität Stuttgart durch. Obwohl es sich um eine quantitative Analyse vergangener Projekte handelte, nicht um eine qualitative Analyse wie in dieser Arbeit, können einige Parallelen zu dieser Arbeit gezogen werden. Die Arbeit hatte unter anderem das Ziel, die Studierenden der nachfolgenden Projekte bei der Aufwandsschätzung zu unterstützen und ihnen einen Vergleich mit Durchschnittswerten aus den Projekten der letzten Jahre zu ermöglichen. Auch diese Arbeit hat zum Ziel, neben der Verbesserung der Qualität der Studienprojekte, auch die Rahmenbedingungen für Studierende zu verbessern. Der Fokus dieser Arbeit liegt dabei jedoch auf der Verbesserung der Betreuung. Hampp fand heraus, dass Umfang und Aufwand von Studienprojekten mit denen von Industrieprojekten vergleichbar sind. Daten zur Qualität der Projekte wurden jedoch nicht erhoben, daher ist nicht klar, ob diese hohe Produktivität auf Kosten der Produktqualität erreicht wird. Insgesamt hält Hampp seine Arbeit geeignet zur Verbesserung der Kostenschätzung von Studienprojekten, jedoch zeigt er auch Probleme auf: zum Beispiel wurden bei der Aufwandserfassung keine einheitlichen Kategorien verwendet, was die Vergleichbarkeit einzelner Projekte verringert.

Auch an der finnischen Tampere University of Technology wurden über mehrere Jahre hinweg während der Durchführung von Studentenprojekten Daten gesammelt, von denen die nachfolgenden Jahrgänge lernen konnten [Aht03]. Die gesammelten Daten beinhalteten aufgetretene Schwierigkeiten und Risiken, Probleme oder Empfehlungen bezüglich Werkzeugen oder Projektmanagement, allgemeine Statistiken sowie Kommentare zu sonstigen erwähnenswerten Themen. Darüber hinaus können die Studierenden ihre aufgewendeten Stunden in den einzelnen Phasen des Projekts mit denen anderer Gruppen vergleichen. Der Autor folgert, dass die Verfügbarkeit der Daten einer der Hauptgründe sei, warum der Kurs als so erfolgreich und

nützlich angesehen werde. Er beschreibt aber auch den hohen Zeitaufwand zur Durchführung dieses Kurses und bringt an, dass der Kurs mit weiterer Klassifikation und Optimierung der gesammelten Daten noch weiter verbessert werden könnte. Auch hier liegt der Fokus wieder auf der direkten Unterstützung der Studierenden bei der Durchführung ihrer Projekte.

An der Eindhoven University of Technology werden ebenfalls seit vielen Jahren Studienprojekte durchgeführt. Poncin, Serebrenik und van den Brand [PSB11] untersuchten im Jahr 2011, wie man aus den Quellcode-Repositories und anderen Datenquellen, wie Issue-Trackern und Mailing-Listen Daten zum Entstehungsprozess der Projekte sammeln kann. Ziel war es Antworten auf einige Fragen, wie die Wiederverwendung der Prototypen oder die Arbeitsverteilung der Studierenden, zu finden. Dies gelang ihnen und sie konnten zeigen, dass sie bei einigen vergangenen Projekten Verstöße gegen Vorgaben früher hätten entdecken können. Zur Analyse der Datenquellen verwendeten Poncin, Serebrenik und van den Brand FRASR und ProM. FRASR [PSV11] ist ein Werkzeug zur Extraktion von *event logs* aus verschiedenen Datenquellen. ProM [VMV+05] kann diese *event logs* mit *data mining*-Methoden analysieren. Das Werkzeug ist ein Vertreter der *process mining*-Werkzeuge.

2.2. Qualitätsmodelle

Qualitätsmodelle bieten die Möglichkeit, Qualitätsanforderungen zu modellieren, die Qualität eines Systems zu analysieren und zu überwachen und passende Metriken dafür zu finden [DJLW09]. Einige Qualitätsmodelle konzentrieren sich vorwiegend auf die Beschreibung von Qualitätseigenschaften, andere bieten zusätzlich eine Methodik, um die Qualität messbar zu machen und Projekte zu vergleichen oder Entwicklungen zu erkennen. Dieser Abschnitt soll einen Einblick in verbreitete Qualitätsmodelle liefern.

2.2.1. ISO 9126

Die ISO 9126 [ISO01] ist ein beschreibendes Qualitätsmodell. Die Norm klassifiziert Qualitätseigenschaften von Software-Systemen. Auf oberster Ebene definiert sie die sechs Merkmale Wartbarkeit, Effizienz, Portabilität, Zuverlässigkeit, Funktionalität und Benutzbarkeit, die durch 28 Teilmerkmale weiter konkretisiert werden (siehe Abbildung 2.1). Eine Quantifizierung der Eigenschaften umfasst die Norm nicht.

2.2.2. ISO 25010

Die ISO 25010 [ISO11] ist ebenfalls ein beschreibendes Qualitätsmodell. Sie ersetzt die ISO 9126 und unterteilt die Software-Qualität in die acht Merkmale Kompatibilität, Portabilität, Funktionalität, Zuverlässigkeit, Gebrauchstauglichkeit, Sicherheit, Effizienz und Wartbarkeit. Jedem dieser Merkmale sind zwei bis sechs Qualitätseigenschaften zugeordnet. Insgesamt

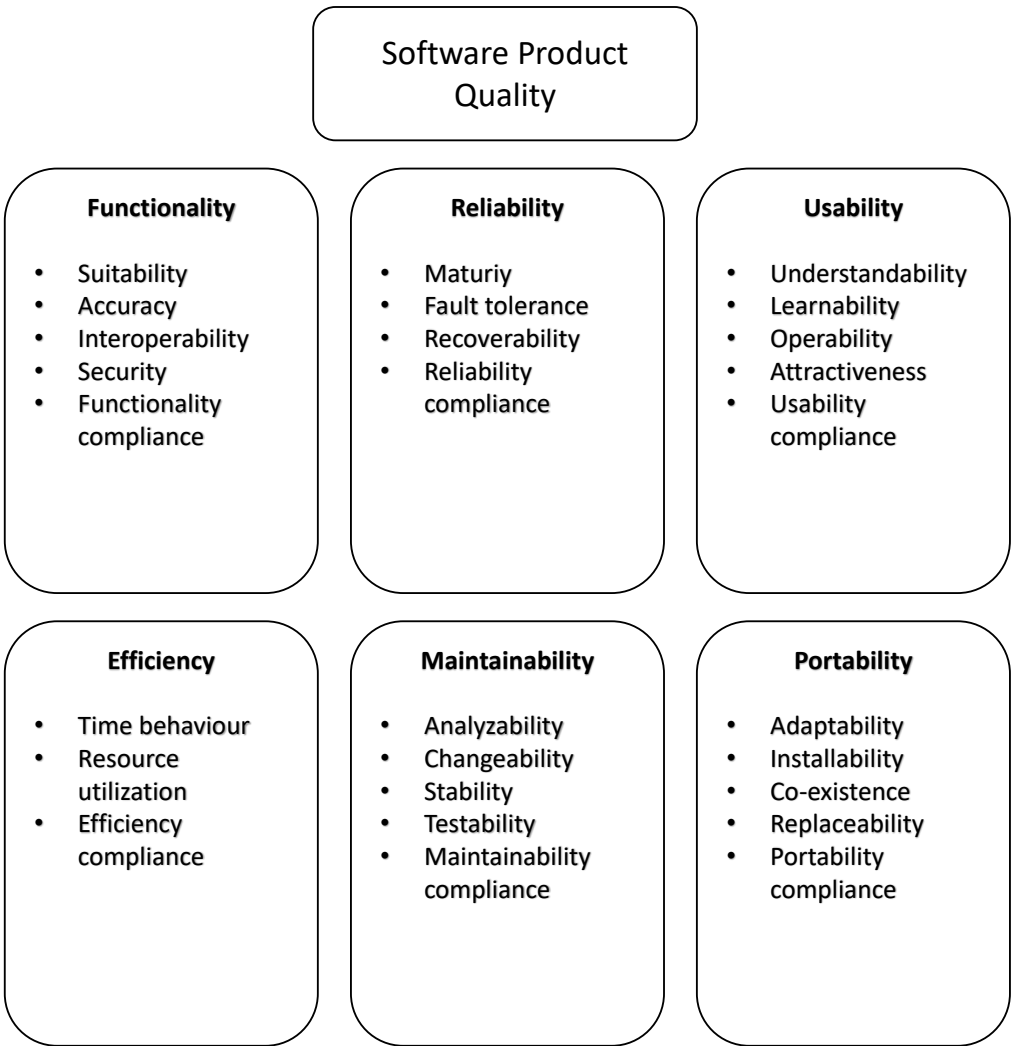

Abbildung 2.1.: Qualitätseigenschaften nach ISO 9126 (eigene Darstellung)

umfasst die Norm 31 Qualitätseigenschaften (siehe Abbildung 2.2). Wie die ISO 9126 umfasst auch diese Norm keine Quantifizierung der Eigenschaften.

2.2.3. Factors-Criteria-Metrics-Methode

Die *Factors-Criteria-Metrics-Methode* kann zur Erstellung von Qualitätsmodellen verwendet werden. Die Methode wurde 1977 von McCall erstmals vorgestellt [CMRW77]. Zur Anwendung der Methode definiert man für den abstrakten Qualitätsbegriff einige Merkmale (engl. „factors“), die einen Aspekt der Qualität genauer beschreiben. Dann werden Teilmerkmale (engl. „criteria“) identifiziert, die die Merkmale genauer definieren. Schließlich werden Metriken (engl. „metrics“) festgelegt, die die Messbarkeit der definierten Teilmerkmale ermöglichen. Ein

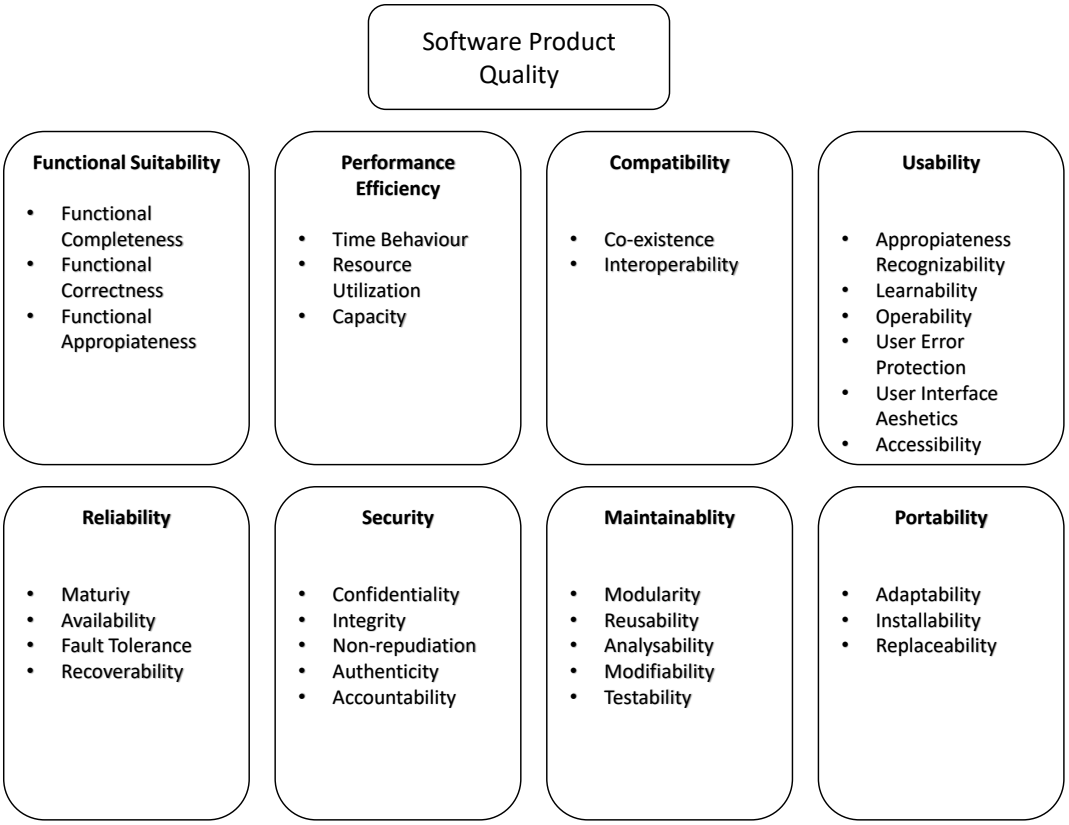

Abbildung 2.2.: Qualitätseigenschaften nach ISO 25010 (eigene Darstellung)

Beispiel für einen solchen Zusammenhang ist das Merkmal *Zuverlässigkeit*, das unter anderem durch das Teilmerkmal *Wiederherstellbarkeit* genauer definiert wird. Eine Metrik zur Messung der *Wiederherstellbarkeit* ist die *Mean Time to Repair (MTTR)*. McCall hat in seiner Arbeit [CMRW77] ein Beispielmodell erarbeitet, für das er elf wichtige Merkmale ermittelte, die von 23 Teilmerkmalen konkretisiert werden. Zur Messung umfasst sein Modell 300 Metriken.

2.2.4. Squale-Modell

Das *Squale-Modell* [MBD+09] basiert auf dem Factors-Criteria-Metrics-Modell von McCall. Der Hauptunterschied besteht in der Einführung einer neuen Ebene der *Practices* zwischen den Ebenen *Criteria* und *Metrics*. Die *Practices* schließen die Abstraktionslücke zwischen diesen Ebenen und ermöglichen dem Betrachter zu erkennen, warum ein Teilmerkmal nicht erfüllt ist und welche Komponenten (Pakete, Klassen, Methoden, …) dafür verantwortlich sind. Beim FCM-Modell nach McCall würde ein schlechter Wert in der Metrik *Mean Time to Repair* zwar auf eine ungenügende *Wiederherstellbarkeit* hinweisen, welche Komponenten nun aber verbessert werden müssen geht daraus nicht hervor. In Squale würde man beispielsweise

für jede Klasse die Häufigkeit von Fehlern und die geänderten Zeilen zur Fehlerbehebung ermitteln und diesen Wert als *Practice* für die *Wiederherstellbarkeit* verwenden.

Squale führt auch ein Konzept zur Berechnung aggregierter Werte aus den einzelnen Messwerten der Metriken ein. In Squale erhält jedes Messobjekt einer *Practice* eine individuelle Note. Diese Note kann über eine diskrete oder stetige Formel berechnet werden. Außerdem erhält jede *Practice* eine systemweite Gesamtnote. Diese wird als gewichteter Durchschnitt aller Individualnoten errechnet. Die Berechnungsformel ist so konstruiert, dass über frei wählbare Konstanten der Einfluss von Ausreißern auf die Gesamtnote bestimmt werden kann. So kann man beispielsweise eine Formel konstruieren, die bereits bei wenigen schlechten Messwerten einen stark negativen Einfluss auf die Gesamtnote hat.

2.2.5. Goal-Question-Metric-Methode

Goal Question Metric (GQM) [CR94] ist eine allgemeine Methode zur Erstellung von Qualitätsmodellen für Software-Systeme. Die GQM-Methode nimmt an, dass man nur in sinnvoller Weise messen kann, wenn man vorher Ziele für seine Organisation oder Projekte definiert hat. Aus diesen Zielen können Metriken und deren Interpretation abgeleitet werden, um das Ziel messbar und vergleichbar zu machen.

Zu diesem Zweck definiert die GQM-Methode drei Ebenen: die konzeptuelle Ebene (Goal), die operationale Ebene (Question) und die quantitative Ebene (Metric). Ein Ziel (engl. „Goal“) bezieht sich immer auf ein Objekt (z. B. Prozess, Produkt, Projekt), hat einen Zweck (z. B. Verbesserung, Überwachung, Vorhersage), setzt einen Qualitätsfokus (z. B. auf eins der ISO 25010 Merkmale), legt den Blickwinkel fest (z. B. Kunde, Entwickler, Projektleiter) und hat einen Kontext (z. B. Projekt, Abteilung, Zeitraum). Zu jedem Ziel werden eine oder mehrere Fragen (engl. „Questions“) definiert, die das untersuchte Objekt charakterisieren und ein spezielles Qualitätsattribut definieren. Zu jeder Frage wird eine Menge an Metriken (engl. „Metrics“) definiert, die die Frage quantifiziert beantworten. Diese Metriken können objektiv sein, d. h. unabhängig vom Blickwinkel und nur abhängig vom untersuchten Objekt oder subjektiv, d. h. abhängig vom Blickwinkel und dem untersuchten Objekt.

Ein GQM-Modell hat eine hierarchische Struktur. An der Spitze steht ein Ziel, das in mehreren Fragen charakterisiert wird. Jede Frage wird wiederum von mehreren Metriken quantifiziert. Eine Metrik kann verwendet werden, um mehrere Fragen zu quantifizieren. Es ist auch möglich, eine Metrik in verschiedenen GQM-Modellen zu verwenden, dann muss jedoch darauf geachtet werden, dass die Daten möglicherweise mehrfach erhoben werden müssen, falls es sich um eine subjektive Metrik handelt.

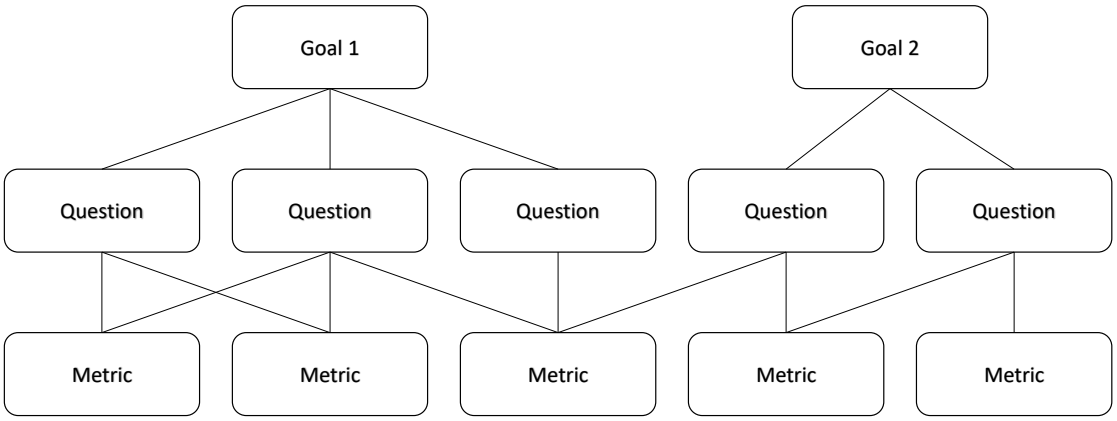

Abbildung 2.3.: Darstellung zweier hierarchischer GQM-Modelle (eigene Darstellung nach [CR94])

2.2.6. Quamoco

Ein weiteres Verfahren zur Modellierung und Bewertung von Software-Qualität ist der *Quamoco*-Ansatz [WLH+12]. *Quamoco* verfolgt, ähnlich wie *Squale*, das Ziel, die Bewertung nach einem Qualitätsmodell direkt in das Modell zu integrieren. Dazu wurde ein Meta-Modell entwickelt, aus dem hierarchische Modelle erzeugt werden können. Das Meta-Modell basiert hauptsächlich auf dem Konzept der *factors*. Es gibt zwei Arten von *factors*, *quality aspects* und *product factors*. Die *quality aspects* repräsentieren abstrakte Qualitätseigenschaften, vergleichbar mit den Eigenschaften aus ISO 9126 oder ISO 25010. Die *product factors* sind messbare Eigenschaften von Komponenten des Systems. Sie können daher immer über eine Metrik berechnet werden. Sowohl *quality aspects* als auch *product factors* können über *sub-aspects* bzw. *sub-factors* konkretisiert werden. *Quality factors* können außerdem von *product factors* beeinflusst werden.

Quamoco definiert darüber hinaus ein Basismodell, das eine Instanz des Meta-Modells ist und auch das Konzept der Module nutzt, das *Quamoco* ebenfalls einführt. Das Basismodell besteht aus drei Modulen: dem objektorientierten Modul und den Modulen für Java und C#, die jeweils vom objektorientierten Modul abgeleitet sind. Für das Basismodell wurde auch eine Bewertungsmethode erarbeitet, die die Aggregation von den Metriken der *product factors* bis zu den *quality factors* erlaubt. Außerdem werden Interpretationsmodelle definiert, die es ermöglichen, die Qualität für Nicht-Entwickler in Schulnoten oder Ampelfarben auszudrücken.

2.2.7. Fazit

Die vorgestellten Qualitätsmodelle unterscheiden sich in ihrer Komplexität und ihren Anwendungsgebieten. Während es sich bei den Normen ISO 9126 und ISO 25010 um einfache

Taxonomien handelt, die nicht direkt auf ein System angewendet werden können, sind die anderen Modelle (*Factors-Criteria-Metrics*, *Squale*, *Goal-Question-Metric* und *Quamoco*) deutlich ausgereifter und können zur Qualitätsbewertung eines Software-Systems verwendet werden. Die Modelle *Quamoco* und *Squale* bieten darüber hinaus Bewertungsmethoden, mit denen die Messwerte der Metriken zu einer systemweiten Gesamtbewertung aggregiert werden können.

2.3. Metriken

Software-Metriken bieten die Möglichkeit, abstrakte Qualitätsmerkmale von Software-Systemen systematisch und quantitativ zu erfassen. Die daraus abgeleiteten Erkenntnisse über die untersuchte Software können dann auf einer objektiven Ebene mit denen anderer Systeme verglichen werden [Hof13].

Ob sich die Erhebung einer Software-Metrik für ein Unternehmen oder eine Organisation lohnt, steht in enger Verbindung mit der Qualität einer Software-Metrik. Hoffmann [Hof13] nennt daher sechs Gütekriterien für Software-Metriken:

Objektivität	Die Messung der Metrik ist frei von subjektiven Einflüssen
Robustheit	Eine wiederholte Messung der gleichen Metrik liefert das gleiche Ergebnis
Vergleichbarkeit	Die Messungen der gleichen Metrik an verschiedenen Produkten sind vergleichbar
Ökonomie	Die Messung einer Metrik kann mit geringen Kosten erfolgen
Korrelation	Die Messerergebnisse einer Metrik lassen Rückschlüsse auf das überwachte Qualitätsmerkmal zu
Verwertbarkeit	Unterschiedliche Messergebnisse haben unterschiedlichen Einfluss auf das zukünftige Handeln

Ludewig und Lichter [LL13] nennen in ihrem Buch Software Engineering sieben Eigenschaften für gute Software-Metriken. Diese sind bis auf eine weitere äquivalent zu denen Hoffmanns:

Verfügbarkeit	Die Bewertung muss dann vorliegen, wenn auf ihrer Grundlage Entscheidungen getroffen werden sollen und nicht erst danach

Im Folgenden wird eine Auswahl wichtiger, verbreiteter und vornehmlich objektorientierter Metriken beschrieben. In Abbildung 2.4 ist eine Einteilung der Metriken in die Kategorien Umfangs-, Kopplungs-, Komplexitäts-, Vererbungs-, Kohäsions- und Dokumentationsmetriken dargestellt, nach denen die Auflistung gruppiert ist. Zusätzlich zu dieser Gliederung können die Metriken auch nach Zugehörigkeit zu einer *metrics suites* (*MOOD metrics suite* oder *ck metrics suite*) klassifiziert werden.

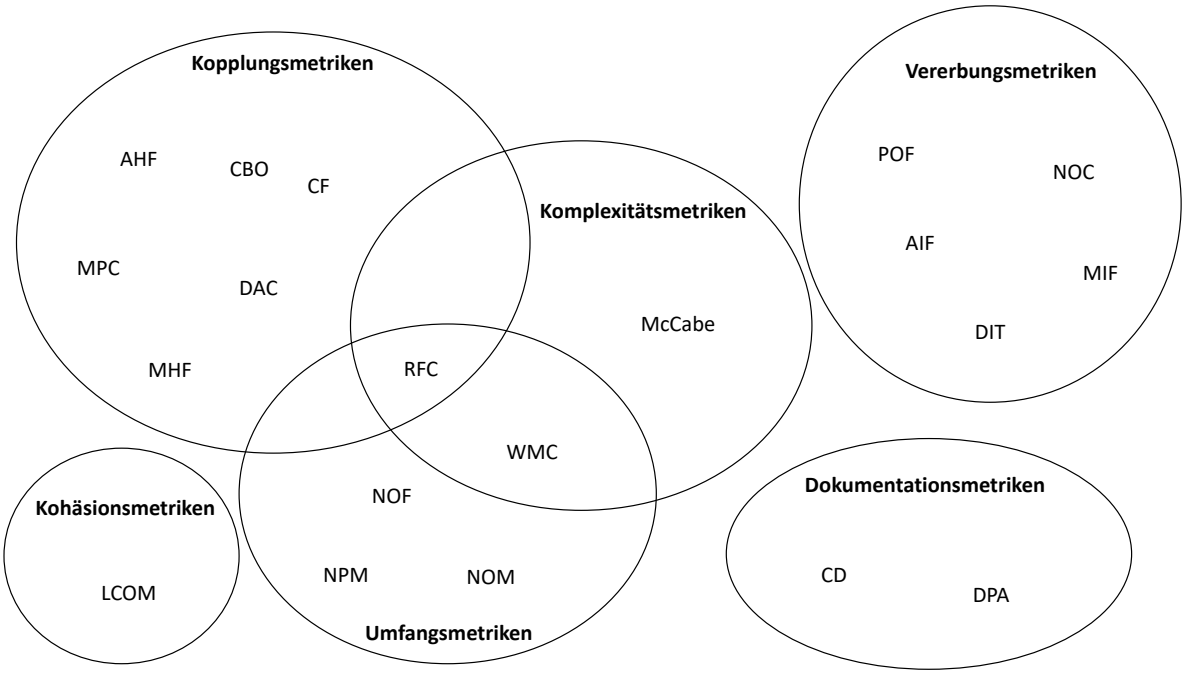

Abbildung 2.4.: Einteilung der vorgestellten Metriken in die Kategorien Umfangsmetriken, Kopplungsmetriken, Komplexitätsmetriken, Vererbungsmetriken, Kohäsionsmetriken und Dokumentationsmetriken

2.3.1. Umfangsmetriken

Umfangsmetriken geben Aufschluss darüber, wie umfangreich der Quelltext eines Systems oder einzelne Komponenten eines Systems sind. Tendenziell kann man sagen, dass die Wartbarkeit von Quelltext abnimmt, je umfangreicher dieser ist.

Number of Fields (NOF) Die Metrik [BV04] zählt die Attribute einer Klasse (sowohl deklarierte, als auch geerbte).

Number of Methods (NOM) Die Metrik [BV04] zählt die Methoden einer Klasse (sowohl deklarierte, als auch geerbte).

Number of Public Methods (NPM) Die Metrik [EHD01] zählt die öffentlichen Methoden einer Klasse (sowohl deklarierte, als auch geerbte).

2.3.2. Kopplungsmetriken

Kopplungsmetriken geben an, wie eng einzelne Komponenten eines Systems miteinander verbunden sind. Viele Methodenaufrufe und referenzierte Klassen sind ein Zeichen für eine hohe Kopplung. Generell sind Systeme, deren Kopplung gering ist, einfacher zu warten, da

bei einer Änderung an einer Stelle nur wenige andere Codestellen berücksichtigt werden müssen.

Method Hiding Factor (MHF) Mithilfe der Kopplungsmetrik [AC94] kann eine Aussage über die Sichtbarkeit von Methoden einer Klasse gemacht werden. Ein Programm mit vielen privaten Methoden hat einen hohen *Method Hiding Factor*. Die Kopplung eines solchen Programms ist geringer als die eines Programms mit weniger privaten Methoden bei ansonsten gleicher Methodenanzahl. Dadurch reduziert sich der Wartungsaufwand des Systems, da viele Codestellen verändert werden können, ohne dass dadurch andere Teile des Systems beeinflusst werden. Die Metrik ist Teil der *MOOD metrics suite*.

Attribute Hiding Factor (AHF) Die Metrik [AC94] gehört zu den Kopplungsmetriken und ermöglicht Aussagen über die Sichtbarkeit von Attributen in einem Programm. Berechnung und Interpretation sind analog zur Metrik *Method Hiding Factor* und die Metrik ist ebenfalls Teil der *MOOD metrics suite*.

Data Abstraction Coupling (DAC) Die Metrik [LH93] misst die Kopplung des Systems auf Datenebene. Sie zählt die Anzahl an verwendeten *Abstract Data Types (ADT)* in einer Klasse. In einer objektorientierten Programmiersprache stellen die Klassen die *ADTs* eines Programms dar. Es werden also die verschiedenen Typen von Attributen und Variablen in einer Klasse gezählt. Auch hier ist eine Klasse mit einem hohen DAC-Wert anfälliger für Änderungen an anderen Klassen. Der Wartungsaufwand steigt.

Coupling Between Objects (CBO) Die Kopplungsmetrik [CK94] zählt für jede Klasse die weiteren Klassen, die sie referenziert. Gezählt werden alle Beziehungen über Vererbung, Methodenaufrufe, Typreferenzen oder Attributreferenzen. Die Metrik ist Teil der *ck metrics suite*.

Message Passing Coupling (MPC) Die Metrik [LH93] zählt die Methoden anderer Klassen, die von den Methoden einer Klasse oder während der Initialisierung der Attribute einer Klasse aufgerufen werden. Eine Methode wird nur einmal gezählt, auch wenn sie öfter aufgerufen wird.

Coupling Factor (CF) Die Metrik [AC94] wird auf Systemebene berechnet und gibt die tatsächliche Anzahl der Beziehungen zwischen Klassen im Verhältnis zur maximal möglichen Anzahl an Beziehungen (jede Klasse nutzt jede andere Klasse) an. Die Beziehungen jeder Klasse werden mithilfe der Metrik *Coupling Between Objects* berechnet. Auch diese Metrik ist Teil der *MOOD metrics suite*.

2.3.3. Komplexitätsmetriken

Komplexitätsmetriken machen eine Aussage über die Struktur von Quellcode. Je höher die Komplexität eines Codeabschnittes, desto schwieriger dessen Wartbarkeit.

McCabe-Metrik Die *McCabe-Metrik* [McC76], auch *zyklomatische Zahl*, gibt die Anzahl der Elementarpfade durch ein Programm an. Die *zyklomatische Zahl* stammt ursprünglich aus der Graphentheorie, wird jedoch meist nicht aus dem Kontrollflussgraphen bestimmt, sondern aus der Anzahl der Verzweigungen im Programm-Code errechnet. Sie ist stets um eins größer, als die Zahl der Verzweigungen im untersuchten Code-Ausschnitt.

2.3.4. Vererbungsmetriken

Vererbungsmetriken geben an, wie stark Konzepte der objektorientierten Vererbung in einem System verwendet werden.

Polymorphism Factor (POF) Die Metrik [AC94] ist in der *MOOD metrics suite* enthalten und gibt die Anzahl der überschriebenen Methoden im Verhältnis zur möglichen Anzahl überschreibbarer Methoden an.

Depth of Inheritance Tree (DIT) Die Metrik [CK94] gibt die Länge des Pfades von einer Klasse bis zu ihrem am weitesten entfernten Vorfahren in der Vererbungshierarchie an. Die Metrik ist Teil der *ck metrics suite*.

Number of Children (NOC) Die Vererbungsmetrik [CK94], die Teil der *ck metrics suite* ist, gibt die Anzahl der Klassen an, die direkt von einer Klasse abgeleitet sind.

Method Inheritance Factor (MIF) Die Metrik [AC94] gibt den Anteil der geerbten Methoden einer Klasse an der Gesamtzahl der Methoden (lokal definierte + geerbte) einer Klasse an. Die Metrik ist Teil der *MOOD metrics suite*.

Attribute Inheritance Factor (AIF) Die Metrik *Attribute Inheritance Factor* [AC94] wird analog zur Metrik *Method Inheritance Factor* berechnet. Die Metrik ist ebenfalls in der *MOOD metrics suite* enthalten.

2.3.5. Kohäsionsmetriken

Kohäsion gibt an, wie gut eine Klasse oder Komponente eine Einheit bildet, die für eine logische Aufgabe zuständig ist. Starke Kohäsion führt zu verbessertet Wartbarkeit und reduzierte Duplizierung von Quellcode.

Lack of Cohesion in Methods (LCOM) Die Kohäsionsmetrik [CK94] gibt an, in wie viele separate Klassen man eine Klasse aufteilen könnte. Zur Berechnung wird ein ungerichteter Graph verwendet. Jeder Knoten repräsentiert eine Methode und zwei Knoten werden verbunden, genau dann, wenn beide ein Attribut der Klasse verwenden, beide eine abstrakte Methode der Klasse verwenden oder die eine Methode die andere Methode aufruft. Ist die Anzahl der unverbundenen Teilgraphen größer als eins, dann sollte diese Klasse in mehrere Klassen aufgeteilt werden. Der Wert der Metrik, die Teil der *ck metrics suite* ist, entspricht der Zahl der Teilgraphen.

2.3.6. Dokumentationsmetriken

Dokumentationsmetriken beschreiben die Menge an Kommentaren und anderer Dokumentation eines Systems. Mangelnde Dokumentation führt zu verringerter Wartbarkeit.

Comment Density (CD) Die Metrik [EHD01] gibt den Anteil der Kommentarzeilen einer Klasse an ihrem Gesamtumfang an.

Documented Public API (DPA) Die Metrik [EHD01] gibt den Anteil der öffentlichen Methoden mit Methodenkommentar an der Gesamtzahl der öffentlichen Methoden einer Klasse an.

2.3.7. Metriken mit mehreren Kategorien

Einige Metriken lassen sich nicht eindeutig in eine Kategorie einordnen, sondern beschreiben verschiedene Aspekte eines Systems.

Weighted Methods per Class (WMC) Die Metrik [CK94] summiert die Komplexitäten der Methoden einer Klasse auf. Zur Ermittlung der Komplexität wird die *McCabe-Metrik* verwendet. Damit macht die Metrik, die Teil der *ck metrics suite* ist, eine Aussage über Umfang und Komplexität einer Klasse.

Response For a Class (RFC) Die Metrik [CK94] gibt an, wie viele weitere Methoden potenziell aufgerufen werden können, wenn eine Methode dieser Klasse aufgerufen wird. Sie wird gebildet, indem zur Anzahl der Methoden der Klasse die Anzahl der Methoden anderer Klassen addiert wird, die in dieser Klasse aufgerufen werden. Die Metrik ist Teil der *ck metrics suite* und erlaubt Aussagen über den Umfang, die Kopplung und die Komplexität eines Programms.

2.3.8. Fazit

Es gibt sehr viele Metriken, die man für objektorientierten Quellcode berechnen kann und noch sehr viele weitere für andere Programmierparadigmen. Eine Metrik erlaubt häufig verschiedene Interpretationen. Aus den in Abschnitt 2.2 vorgestellten Qualitätsmodellen geht hervor, welche Metriken man zur Messung eines Systems verwenden sollte und wie diese interpretiert werden kann. Ein reines Berechnen von Metriken ohne dazugehöriges Qualitätsmodell ist daher nicht sinnvoll.

2.4. Werkzeuge

Zur Erhebung von Metriken und anderen Daten der statischen Codeanalyse gibt es eine Vielzahl an kommerziellen und frei verfügbaren Werkzeugen. Die Werkzeuge unterscheiden sich teilweise deutlich in ihrer Funktionalität und potenziellen Anwendungsbereichen. Der folgende Abschnitt liefert eine Klassifikation der Werkzeuge in die Kategorien freie Werkzeuge, IDE-Plugins, kommerzielle Werkzeuge und Werkzeuge zur statischen Codeanalyse. Zu jeder Kategorie werden einige Vertreter genauer beschrieben. Die Auswahl der Werkzeuge beschränkt sich auf solche, mit denen Java-Programme analysiert werden können, da die für diese Arbeit untersuchten Projekte durchgängig in Java entwickelt sind.

2.4.1. Freie Werkzeuge

Hier sind mit *freie Werkzeuge* alle kostenlosen Werkzeuge gemeint, die nicht als Erweiterung in ein anderes Programm integriert werden und zur Metrikberechnung verwendet werden können. Bei freien Werkzeugen handelt es sich meist um *Open-Source-Programme*, die von einer Entwicklergruppe oder Organisation entwickelt werden oder um die kostenfreie Version eines ansonsten kostenpflichtigen, kommerziellen Tools.

SonarQube [Son] ist ein *Open-Source-Werkzeug*. *SonarQube* wird von *SonarSource* entwickelt und ist unter der LGPL-Lizenz veröffentlicht. Das Werkzeug bietet eine Plattform zur statischen Codeanalyse für die Programmiersprachen Java, Groovy, Flex, PHP, PL/SQL, C#, Cobol, .NET und Visual Basic 6. Der Fokus von *SonarQube* liegt auf der regelbasierten Ermittlung von potenziellen Fehlern und Schwachstellen, aber auch duplizierter Code und Verstöße gegen Kodierrichtlinien werden gefunden. Metriken berechnet *SonarQube* hauptsächlich im Bereich Komplexität und Dokumentation. Da die Plattform allerdings über einen Plugin-Mechanismus verfügt, ist es möglich, Erweiterungen einzubinden, die zusätzliche Metriken berechnen oder andere Werkzeuge zur Berechnung integrieren.

SourceMeter ist ein kommerzielles Werkzeug, von dem eine umfangreiche kostenlose Variante erhältlich ist. *SourceMeter* [Fro] kann zur statischen Codeanalyse für die Programmiersprachen C/C++, Java, C#, Python und RPG verwendet werden. Die Java-Version des Werkzeugs berechnet eine Reihe von Codemetriken auf unterschiedlichen Ebenen (Paket, Klasse oder Methode), weist auf Sicherheitslücken und Programmierfehler hin, findet duplizierten Code und bietet die Möglichkeit der Integration der bekannten Analysewerkzeuge *FindBugs* und *PMD*. Die meisten der Funktionen sind in der freien Version des Werkzeugs enthalten, für einige muss jedoch eine Lizenz für die kostenpflichtigen Basic oder Pro Versionen erworben werden.

2.4.2. Kommerzielle Werkzeuge

Gerade im Bereich der Java-Entwicklung gibt es viele kommerzielle Werkzeuge zur Berechnung von Metriken. Aus Kostengründen können diese Programme in dieser Arbeit nicht für die Analyse verwendet werden, nichtsdestotrotz können sie für andere Projekte von Relevanz sein.

Mit *Resource Standard Metrics* [M S] können Metriken eines Programm berechnet werden. Außerdem ermittelt das Werkzeug potenzielle Fehler im Code.

JHawk [Vir] kann 106 Metriken auf Java-Programmen berechnen. Die Analyse kann über die Kommandozeile gestartet werden und es stehen viele verschiedene Möglichkeiten zum Export der Ergebnisse zur Verfügung.

Imagix 4D [Ima] ist ein Werkzeug, das beim Verstehen und Analysieren eines fremden Programms hilft. Eher als Nebenprodukt berechnet das Tool auch Metriken des untersuchten Systems.

Mit *Essential Metrics* [Pow] können Softwaremetriken von C/C++ und Java-Programmen ermittelt werden. Das Programm kann über die Kommandozeile ausgeführt werden, wird allerdings seit zwei Jahren nicht mehr gewartet.

Java Source Code Metrics [Sem] berechnet einige wenige Metriken von Java-Programmen. Selbst einige freie Werkzeuge unterstützen mehr Metriken.

2.4.3. IDE-Plugins

Ähnlich zum Plugin-Mechanismus von *SonarQube* ist es möglich mithilfe von Plugins den Funktionsumfang moderner Entwicklungsumgebungen zu erweitern. Zu den Funktionen zählen neben der Unterstützung weitere Programmiersprachen oder Dateiformate auch Plugins zur statischen Codeanalyse. Für die Entwicklungsumgebungen *IntelliJ IDEA* und *Eclipse* sind Erweiterungen verfügbar, mit denen Metriken des aktuellen Projekts berechnet werden können. *IntelliJ IDEA* kann um das Plugin *MetricsReloaded* [Bas] ergänzt werden. Für *Eclipse* gibt es mit *Metrics* [sau] und *Eclipse Metrics Plugin* [Wal] gleich zwei Erweiterungen zur Berechnung von Codemetriken. Die Plugins *Metrics* und *MetricsReloaded* lassen sich zudem über die Kommandozeile ausführen, ohne die ohne die Entwicklungsumgebung starten zu müssen.

2.4.4. Werkzeuge zur statischen Codeanalyse

Neben Werkzeugen, die auf die Berechnung von Codemetriken spezialisiert sind, gibt es im Java-Umfeld einige sehr bekannte Werkzeuge zur statischen Codeanalyse. Teilweise können die Ergebnisse dieser Werkzeuge auch als Metriken verwendet werden, zum Beispiel kann die Anzahl der leeren Catch-Blöcke als Metrik zur Bestimmung der Resilienz angesehen werden.

FindBugs [Fin] ist ein freies Werkzeug, das im Java-Bytecode nach Fehlermustern sucht, die häufig Anzeichen für wirkliche Fehler sind. Das Tool wird von der *University of Maryland* entwickelt und ist unter der LGPL-Lizenz verfügbar. *FindBugs* kann über eine eigene Benutzeroberfläche, die Kommandozeile oder als Plugin in anderen Systemen (z. B. gibt es auch ein *FindBugs*-Plugin für *SonarQube*) verwendet werden.

PMD [PMD] ist ein freies Analysetool, das vor allem *Bad Practices* im Quellcode aufspürt, die nicht direkt zu einem Fehler führen, sondern einen anderen negativen Einfluss auf den Quelltext haben, zum Beispiel eine verschlechterte Lesbarkeit. *PMD* kann als Plugin in viele IDEs und Build-Tools integriert und genutzt werden.

Checkstyle [che] ist ein freies Werkzeug, das die Einhaltung von Kodierrichtlinien überprüft und entsprechende Verstöße anzeigt. Das Tool kann als eigenes Java-Programm oder mithilfe eines der vielen Plugins für IDEs und Build-Tools ausgeführt werden.

2.4.5. Fazit

Im Java-Umfeld gibt es sehr viele Werkzeuge zur statischen Codeanalyse und Berechnung von Metriken. Die Tools unterscheiden sich in ihrem Einsatzgebiet: Es gibt Tools zur Unterstützung der Entwickler (z. B. IDE-Plugins), zur grafischen Darstellung von Messungen, zum Beispiel für die Projektleitung (z. B. *SonarQube*) und zur Berechnung von Metriken und anderen Kennzahlen (z. B. *SourceMeter*). Je nach Verwendungszweck muss daher ein passendes Werkzeug ausgewählt werden oder es müssen mehrere Werkzeuge kombiniert werden.

3. Methodik

Im Rahmen der vorliegenden Arbeit werden die Studienprojekte anhand von drei Analysen untersucht. Die Ziele und Vorgehensweisen dieser drei Analysen werden in diesem Kapitel dargelegt. Die Analysen dienen der Erhebung von Daten zur Ableitung von Handlungsempfehlungen, welche die Betreuung und die Qualität der Studienprojekte nachhaltig verbessern sollen. In Abbildung 3.1 ist der Ablauf der drei Analysen grafisch dargestellt. In Kapitel 4 wird eine Implementierung zur Durchführung der beiden ersten Analysen vorgestellt. Die dritte Analyse wird manuell durchgeführt. Die Ergebnisse aller drei Analysen werden in Kapitel 5 behandelt.

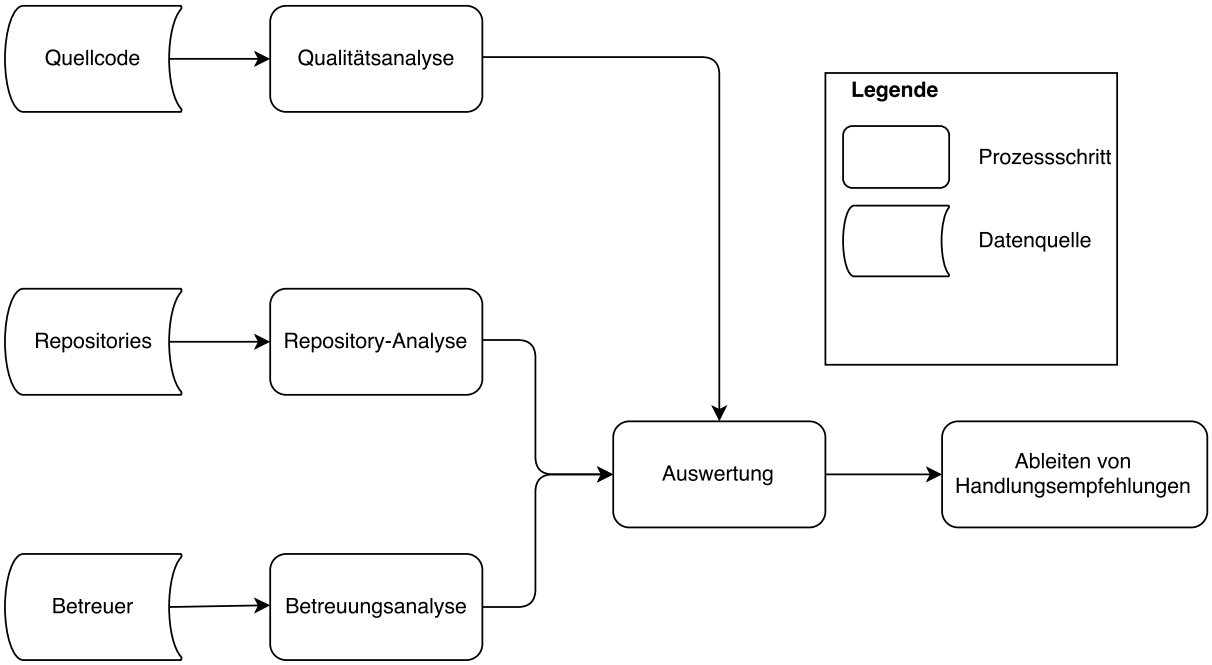

Abbildung 3.1.: Ablaufdiagramm der Analysen der Methodik

3.1. Qualitätsanalyse

3.1.1. Ziel der Analyse

Ziel der Qualitätsanalyse ist es, die Studienprojekte der vergangenen Jahre auf eine objektive Weise miteinander vergleichen zu können. Es soll eine Benchmark geschaffen werden, auf deren Skala die vergangenen und möglichen zukünftigen Projekte eingeordnet werden können. Die Analyse muss daher die Qualität der Projekte auf einen einzelnen skalaren Wert reduzieren. Daraufhin wird mithilfe der zwei weiteren Analysen untersucht, was potenzielle Einflussfaktoren auf die entstandene Benchmark sind und wie man daraus für die Zukunft lernen kann.

3.1.2. Durchführung der Analyse

Die Qualitätsanalyse wird mittels der GQM-Methode (siehe Abschnitt 2.2.5) durchgeführt, da es sich bei dieser Methode um einen universellen und einfach umzusetzenden Ansatz handelt. Die Definition von Zielen hilft, strukturiert vorzugehen und sich auf die grundlegenden Aspekte zu konzentrieren. Die Qualitätsmodelle *Quamoco* und *Squale* (siehe Abschnitte 2.2.6 und 2.2.4) könnten ebenfalls für diese Analyse verwendet werden. Da es sich bei dieser Arbeit um eine explorative Studie mit einer geringen Anzahl an untersuchten Projekten handelt und daher keine Allgemeingültigkeit der Ergebnisse erwartet werden kann, wurde für diese Analyse die GQM-Methode als pragmatischer Ansatz ausgewählt.

Formulierung des GQM-Modells

Die vorgestellte Methodik orientiert sich am Qualitätsbegriff der ISO 25010 (siehe Abschnitt 2.2.2). Im Rahmen der vorliegenden Arbeit soll eines der Merkmale der Norm analysiert werden. Ausgewählt wird das Merkmal *Wartbarkeit*, da dies mit typischen Code-Metriken am besten zu quantifizieren ist und da für die weitere Nutzung der Projekte durch das Fraunhofer IAO vor allem die Teilmerkmale der Wartbarkeit von Bedeutung sind.
Das Ziel des GQM-Modells ist daher die Analyse der Wartbarkeit. Die Fragen zur Charakterisierung des Ziels orientieren sich an den fünf Teilmerkmalen[1] des Merkmals Wartbarkeit der ISO 25010 und lauten:

- *Wie stark ist der Quelltext modularisiert?*
- *Wie gut kann der Quelltext wiederverwendet werden?*
- *Wie gut ist der Quelltext analysierbar?*

[1] Modularisierung, Wiederverwendbarkeit, Analysierbarkeit, Modifizierbarkeit, Testbarkeit

- *Wie gut ist der Quelltext modifizierbar?*
- *Wie gut kann der Quelltext getestet werden?*

Ostberg und Wagner [OW14] führen an, dass viele ihrer Industriepartner und Studierenden Metriken zur Quantifizierung von Wartbarkeit verwenden, die veraltet sind oder deren Zusammenhang zur Wartbarkeit unklar ist. Bei der Auswahl der quantifizierenden Metriken ist es daher wichtig, nur solche Metriken auszuwählen, deren Zusammenhang mit dem quantifizierten Qualitätsmerkmal empirisch oder wissenschaftlich argumentierend belegt ist. Die folgende Auflistung verzeichnet zu jedem Teilmerkmal diejenigen Metriken, zu denen wissenschaftliche Arbeiten eine solche Korrelation belegen.

Modularisierung	Response for Class [BDW99], Coupling between Objects [BDW99], Message Passing Coupling [BDW99], Data Abstraction Coupling [BDW99], Coupling Factor [BDW99]
Wiederverwendbarkeit	Weighted Methods per Class [EHD01], Lack of Cohesion of Methods [EHD01], Number of Public Methods [EHD01], Comment Density [EHD01], Documented Public API [EHD01]
Analysierbarkeit	Weighted Methods per Class [Zus93], Depth of Inheritance Tree [HCN00], Coupling Factor [Mey88]
Modifizierbarkeit	Number of Methods [LH93], Weighted Methods per Class [LH93], Response for Class [LH93], Coupling between Objects [CDK98], Lack of Cohesion of Methods [BV04; CDK98], Depth of Inheritance Tree [CDK98; HCN00], Number of Children [BV04], Message Passing Coupling [BV04], Data Abstraction Coupling [BV04], Methods Inheritance Factor [AM96], Attributes Inheritance Factor [AM96], Polymorphism Factor [AM96], Coupling Factor [AM96]
Testbarkeit	Number of Fields [BV04], Number of Methods [BV04], Weighted Methods per Class [BV04], Response for Class [BV04]

Berechnung der Benchmark

Die Qualitätsmodelle *Quamoco* und *Squale* (siehe Abschnitte 2.2.6 und 2.2.4) enthalten spezielle Ansätze zur Aggregation von Metrikwerten, um die Gesamtbewertung eines Projekts vornehmen zu können. Die *GQM-Methode* enthält kein solches Verfahren. Daher ist die Erstellung eines individuellen Vorgehens nötig. Alternativ könnten die Methoden der Modelle *Quamoco* und *Squale* für das *GQM-Modell* angepasst werden. Aus denselben Gründen wie in Abschnitt 3.1.2 wird für diese Arbeit aber ein eigenes, pragmatisches und unkompliziertes Verfahren entwickelt. Das in dieser Arbeit verwendete Vorgehen erfolgt in vier Schritten:

1. Für jedes Projekt werden die Metriken des GQM-Modells berechnet.
2. Für jede Metrik wird das Projekt mit dem besten Wert bestimmt. Alle Projekte erhalten als Note für eine Metrik den Quotienten aus ihrem Metrikwert und dem Metrikwert des besten Projekts. Dadurch erhält jedes Projekt eine Note zwischen 0 und 1.
3. Die Noten eines Projekts werden für jede Frage des GQM-Modells aufsummiert. Nun wird wieder das Projekt mit der besten Bewertung ermittelt und die anderen Projekte werden dazu in Relation gesetzt. Daher liegt auch die Note einer Frage immer zwischen 0 und 1.
4. Die Gesamtnote für das Ziel des GQM-Modells (Wartbarkeit) errechnet sich als Summe der Noten für die einzelnen Fragen. Da das GQM-Modell aus fünf Fragen besteht, liegt die Gesamtnote für jedes Projekt zwischen 0 und 5.

3.2. Repository-Analyse

3.2.1. Ziel der Analyse

Diese Arbeit basiert auf der Annahme, dass der Entstehungsprozess der Studienprojekte einen Einfluss auf die Ergebnisse und die Qualität der Projekte hat. Um dies zu untersuchen, werden Daten zum Entstehungsprozess aus den Quellcode-Repositories der betrachteten Projekte ermittelt. Diese Informationen können anschließend den Ergebnissen der Qualitätsanalyse gegenübergestellt werden, um mögliche Zusammenhänge festzustellen.

3.2.2. Durchführung der Analyse

Neben dem eigentlichen Versionsverlauf mit Commits und geänderten Dateien und Zeilen können aus den Quellcode-Repositories weitere Metadaten zu jeder Version erhoben werden. Daten wie der Autor, der Zeitpunkt des Commits, die Commit-Nachricht und die geänderten Zeilen können zusammen mit den Commits abgerufen werden. Diese Metadaten werden in

der Repository-Analyse erhoben, um daraus Metriken zum Vergleich der Projekte zu entwickeln. Zur Ableitung der Metriken können die Daten gruppiert oder summiert werden und Durchschnitte gebildet werden.

Für diese Arbeit wurden die folgenden Fragen anhand von Analysen der Quellcode-Repositories beantwortet:

- Sind die Commits gleichmäßig auf alle Tage verteilt oder gibt es Tage, die deutlich vom Durchschnitt abweichen?
- Sind die geänderten Zeilen gleichmäßig auf alle Tage verteilt oder gibt es Tage, die deutlich vom Durchschnitt abweichen?
- Sind die geänderten Zeilen gleichmäßig auf alle Teilnehmer verteilt oder gibt es Studierende, deren Arbeitsanteil deutlich vom Durchschnitt abweicht?
- Sind die Commits gleichmäßig auf alle Wochentage verteilt oder gibt es Wochentage, die deutlich vom Durchschnitt abweichen?
- Wie hoch ist der Anteil der Commits, die am Wochenende gemacht wurden?
- Sind die Commits gleichmäßig auf alle Studierenden verteilt oder gibt es Studierende, deren Arbeitsanteil deutlich vom Durchschnitt abweicht?
- Werden pro Commit in etwa gleich viele Zeilen geändert oder gibt es Commits, die deutlich vom Durchschnitt abweichen?

Die Fragen sind teilweise aus den charakteristischen Eigenschaften der Studienprojekte abgeleitet, die in Abschnitt 1.1 erwähnt werden. Die Fragen zur Verteilung der Commits nach Kalendertagen und Wochentagen unterstellen, dass es Projekte gibt, die den Hauptteil der Arbeit kurz vor den vereinbarten Präsentationsterminen erledigen bzw. nur an den vereinbarten Arbeitstagen vor Ort arbeiten.

Um die Gleichmäßigkeit der Verteilungen bzw. die Abweichungen der Messwerte vom Durchschnitt zu berechnen, kann die Standardabweichung verwendet werden. Da sich die Mittelwerte der Verteilungen stark unterscheiden können und daher damit zu rechnen ist, dass die Standardabweichungen ebenfalls stark schwanken (eine Verteilung mit einem hohen Mittelwert hat häufig auch eine höhere Standardabweichung und umgekehrt), wird statt der Standardabweichung der Variationskoeffizient verwendet, der als Quotient aus Standardabweichung und Mittelwert definiert ist.

Um ein einheitliches Vorgehen für alle Repositories zu gewährleisten, wird im Kontext dieser Arbeit vorausgesetzt, dass es sich bei den untersuchten Repositories um *Git-Repositories* handelt. Da es sich bei den meisten der untersuchten Repositories um *SVN-Repositories* handelt, wird im Zuge der Analyse als erster Schritt eine Konvertierung zu Git mithilfe des Werkzeugs *git-svn* [Git] durchgeführt.

3.3. Betreuungsanalyse

3.3.1. Ziel der Analyse

Die Betreuungsanalyse verfolgt zwei Ziele: Zum einen soll sie Informationen über die Betreuung der Studienprojekte erheben, die potenziell einen Einfluss auf die Qualität der Projekte haben, zum anderen sollen Daten erhoben werden, die zur Validierung der Qualitätsanalyse aus Abschnitt 3.1 verwendet werden können. Die Informationen zur Betreuung können anschließend auf Zusammenhänge mit den Ergebnissen der Qualitätsanalyse hin geprüft werden.

3.3.2. Durchführung der Analyse

Zur Erreichung der Ziele wurde eine Befragung der Betreuer der untersuchten Projekte durchgeführt. Der Fragebogen enthält allgemeine Fragen zum Projekt, Fragen zur Qualität, zur Betreuung und zur Umsetzung. Die allgemeinen Fragen dienen hauptsächlich dazu, einen besseren Einblick in die untersuchten Projekte zu erhalten. Zum Beispiel soll eine Einschätzung des subjektiven Erfolgs des Projekts abgegeben werden oder die Komplexität des technischen und fachlichen Hintergrunds beurteilt werden. Die Fragen zur Qualität haben zum Ziel, die Qualitätsanalyse aus Abschnitt 3.1 zu validieren, indem die Einschätzung der Betreuer mit den Ergebnissen der Analyse verglichen wird. Aus den Fragen zur Betreuung und der Umsetzung sollten Rückschlüsse auf den Betreuungsprozess gezogen werden, die wiederum zu den Ergebnissen der Qualitätsanalyse in Relation gesetzt werden können. Dadurch kann ermittelt werden, welchen Einfluss die Betreuung und der Betreuungsprozess auf die Qualität eines Studienprojekts hat.

Der Fragebogen wurde als Word-Dokument per E-Mail mit einem persönlichen Anschreiben verschickt. Auf ein Online-Umfrage-Tool wurde aufgrund der geringen Teilnehmerzahl und der weniger persönlichen Ansprache verzichtet. Im Anhang A.2 ist der vollständige Fragebogen abgebildet, wie er an die Betreuer der untersuchten Projekte verschickt wurde.

3.4. Auswertung

Zur Auswertung der Ergebnisse wird ermittelt, ob die Ergebnisse der Repository-Analyse und der Betreuungsanalyse einen messbaren Einfluss auf die Benchmark aus der Qualitätsanalyse haben. Dazu werden Korrelationen mit der Benchmark mithilfe des Korrelationskoeffizienten nach Pearson [Pea95] berechnet. Je höher dieser Koeffizient r ist, desto stärker ist der lineare Zusammenhang zwischen den untersuchten Aspekten.

Um eine Korrelation berechnen zu können, muss für jedes Studienprojekt genau ein Benchmarkwert und genau ein Wert für jeden untersuchten Aspekt vorliegen. Um die Korrelationen der Benchmark mit den Verteilungen aus der Repository-Analyse zu berechnen, wird der Variationskoeffizient verwendet. Die am Wochenende erstellten Commits werden einfach als Anteil der gesamten Commits berechnet. Dieser Quotient kann dann zur Korrelationsberechnung verwendet werden.
Zur Berechnung der Korrelationen der Betreuungsanalyse werden die Antwortmöglichkeiten der Ordinalskalen von eins bis vier nummeriert. Da der Durchschnitt auf Ordinalskalen nicht angewendet werden kann, wird der Median als Grundlage der Berechnung des Korrelationskoeffizienten verwendet.

4. Implementierung

4.1. Anforderungen

Zur Durchführung der in Kapitel 3 beschriebenen Methodik wurde ein Werkzeug entwickelt, mit dem die Qualitätsanalyse (Berechnung der Metriken, Aggregation und Berechnung der Benchmark) und die Analyse der Quellcode-Repositories (Extraktion der Daten aus den Commits) durchgeführt werden können. Dabei wurden die folgenden Anforderungen berücksichtigt, die sich aus der Ausschreibung und Aufgabenstellung dieser Arbeit ableiten:

- Die Analyse soll möglichst automatisiert durchgeführt werden können. Manuelle Schritte sind möglich, wenn es einen pragmatischen Grund hierfür gibt.
- Zur Berechnung der Metriken sollen bestehende Werkzeuge verwendet werden, da es hierfür bereits geeignete Werkzeuge gibt (vgl. Abschnitt 2.4).
- Die Ergebnisse der Analyse sollen mittels Web-Frontend zusammengeführt und präsentiert werden.
- Aus dem Web-Frontend soll erkennbar sein, wie gut ein Projekt im Vergleich zu allen anderen untersuchten Projekten abschneidet.
- Das Web-Frontend soll für jedes Projekt erstellt werden können, um den Teilnehmern von laufenden Projekten einen Vergleich mit anderen Projekten zu ermöglichen.

4.2. Resultat

Das Werkzeug wird in zwei unabhängigen Teilen entwickelt: dem Analysewerkzeug und dem Web-Frontend. Das Analysewerkzeug berechnet Metriken der untersuchten Projekte mithilfe des Werkzeugs *SourceMeter* (siehe Abschnitt 2.4.1). Das Tool berechnet eine große Anzahl an Metriken auf verschiedenen Abstraktionsebenen (Paket, Klasse oder Methode). Außerdem lässt sich das Werkzeug über die Kommandozeile ausführen und kann daher leicht in eigene Programme integriert werden. Die Einschränkungen der freien Version im Vergleich zur kostenpflichtigen Basic- oder Pro-Version sind im Kontext dieser Arbeit nicht relevant, da alle benötigten Funktionen ebenfalls in der freien Version enthalten sind.
Darüber hinaus aggregiert das Analysewerkzeug die Metriken anhand des GQM-Modells (siehe

Abschnitt 3.1) und berechnet die Benchmark. Der Export der generierten Daten erfolgt in .csv-Dateien, die vom Web-Frontend eingelesen werden.
Die Repository-Analyse wird ebenfalls vom Analysewerkzeug durchgeführt. Dafür wird eine Repräsentation des Projekts als Git-Repository vorausgesetzt. Die Konvertierung von SVN-Repositories erfolgte mit dem Werkzeug *git-svn* [Git]. Die Ergebnisse der Analyse werden ebenfalls im .csv-Format exportiert und vom Web-Frontend eingelesen.

Das Analysewerkzeug wird in der Programmiersprache *Python* entwickelt. *Python* ist eine leichtgewichtige Skriptsprache und unterstützt mehrere Programmierparadigmen. Das entstandene Werkzeug wurde hauptsächlich im objektorientierten Programmierparadigma implementiert. Für *Python* spricht, dass viele Standardbibliotheken enthalten sind und weitere Bibliotheken mit geringem Aufwand nachinstalliert werden können. Außerdem können die entstehenden Programme bis zu einer gewissen Größe schnell entwickelt werden und Änderungen am Programm können leicht vorgenommen werden. Dies unterstützt die iterative Entwicklung der Methodik der vorliegenden Arbeit.

Für jedes untersuchte Projekt wird ein separates Web-Frontend erstellt, das aus zwei Seiten besteht. Auf der einen Seite kann sich der Nutzer die Ergebnisse der Qualitätsanalyse für das untersuchte Projekt ansehen (siehe Abbildung 4.1). Dort werden die Werte der Benchmark, der Teilmerkmale und der einzelnen Metriken angezeigt. Ein farbiger Punkt in der Ecke gibt an, ob sich das Projekt damit im oberen, mittleren oder unteren Drittel der untersuchten Projekte befindet. Über einen Klick auf eine der Boxen öffnet sich ein Dialog, der weitere Informationen, wie den vollständigen Namen der Metrik anzeigt.

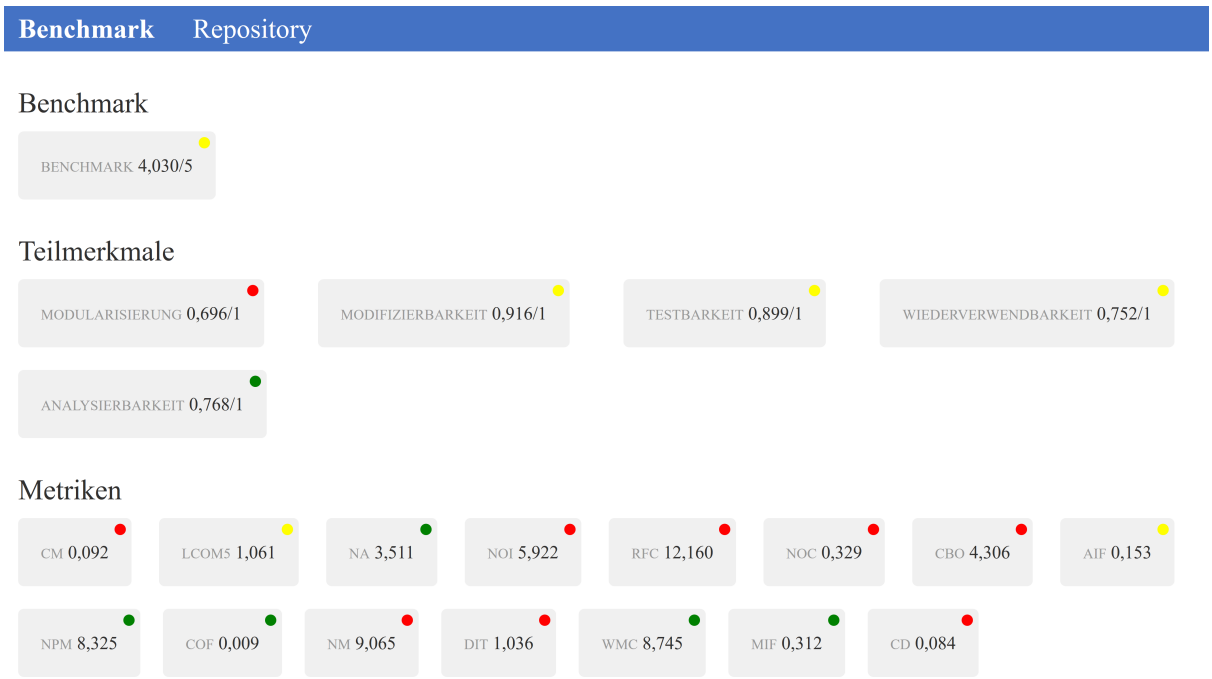

Abbildung 4.1.: Ergebnisse der Qualitätsanalyse (Beispielansicht)

Die zweite Seite zeigt drei Diagramme der Repository-Analyse (siehe Abbildung 4.2). Die Diagramme visualisieren die Daten der Analyse, für die ein Zusammenhang mit den Ergebnissen der Benchmark ermittelt werden konnte (siehe Abschnitt 5.2).

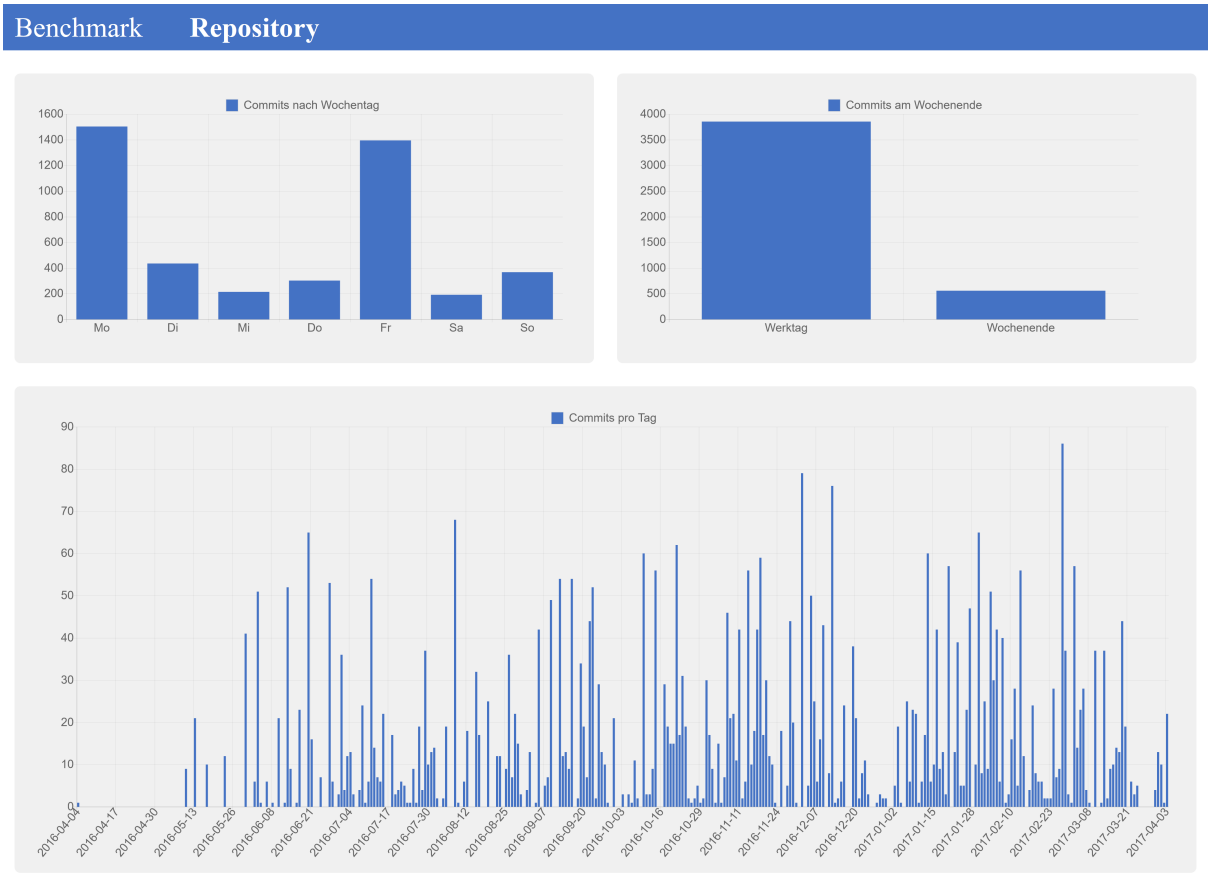

Abbildung 4.2.: Ergebnisse der Repository-Analyse (Beispielansicht)

Das Web-Frontend wird mit dem Web-Framework *Angular* in der Programmiersprache *TypeScript* umgesetzt. Die Architektur einer *Angular*-Anwendung besteht aus hierarchischen Komponenten. *Angular* bietet Dependency Injection, Routing und es existieren viele Drittanbieter-Bibliotheken. *TypeScript* ist eine Obermenge der Programmiersprache *JavaScript* und wird von *Microsoft* entwickelt. Die Sprache unterstützt objektorientierte Klassen und Vererbung und ermöglicht im Gegensatz zu *JavaScript* die Typisierung von Variablen und Methoden.

Web-Frontend und Analysewerkzeug bilden jeweils eine Komponente (siehe Komponentendiagramm in Abbildung 4.3). Das Web-Frontend besteht aus vier Angular-Komponenten: eine Basiskomponente, je eine Komponente für die beiden Webseiten und eine Komponente, um die Detailinformationen in einem modalen Dialog anzuzeigen. Außerdem enthält es zwei *Services* zum Einlesen und Aufbereiten der Daten.
Das Analysewerkzeug besteht aus sieben Python-Klassen, von denen fünf jeweils einen Schritt

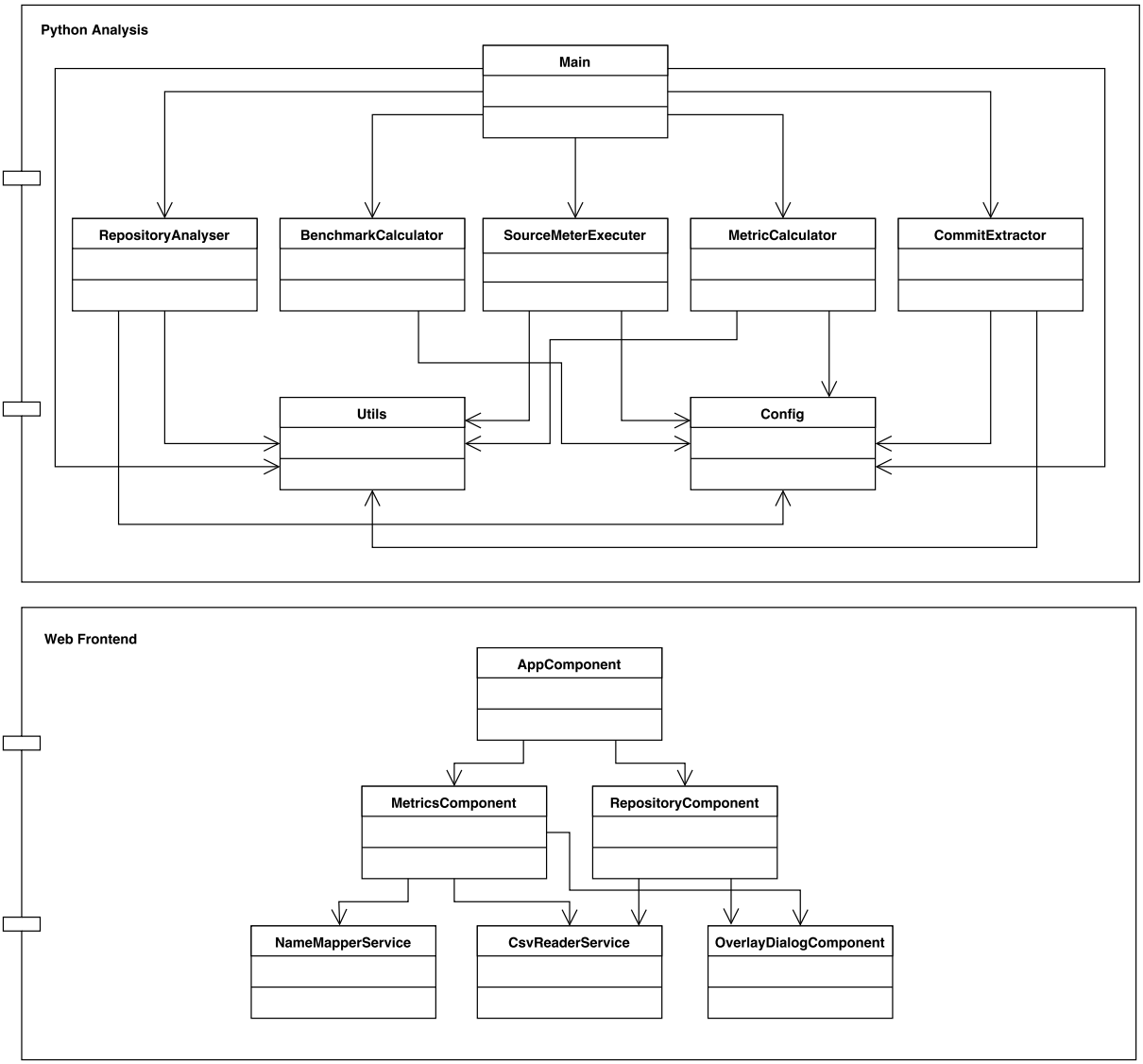

Abbildung 4.3.: Komponentendiagramm des entstandenen Werkzeugs

der Analyse durchführen. Außerdem gibt es eine Klasse für allgemeine Funktionalitäten und eine für die Konfiguration. Die Qualitätsanalyse und die Repository-Analyse werden unabhängig voneinander durchgeführt (siehe Ablaufdiagramm in Abbildung 4.4). Aus den Ergebnissen der Analyse wird anschließend eine lauffähige Version des Web-Frontends für jedes Projekt generiert, das nur die Daten dieses einen Projekts enthält und daher auch an Dritte weitergegeben werden kann.

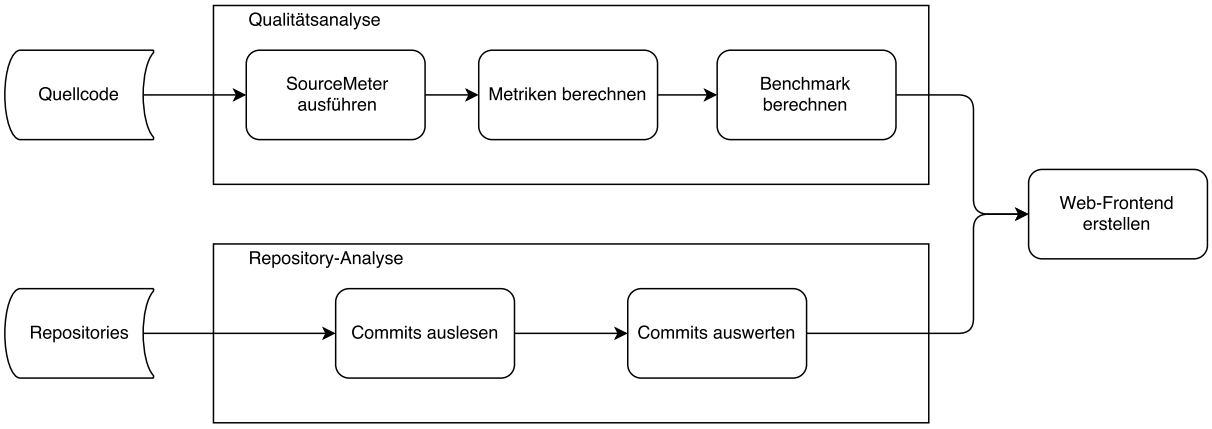

Abbildung 4.4.: Ablaufdiagramm des entstandenen Werkzeugs

4.3. Verwendung des Werkzeugs

Das Web-Frontend gibt dem Nutzer des Werkzeugs einen Überblick über den Leistungsstand des betrachteten Projekts im Vergleich zu allen untersuchten Projekten. Außer zur besseren Visualisierung wurde das Web-Frontend für die Analysen dieser Arbeit nicht verwendet. Zur Durchführung der Analysen wurden jedoch auch die Ausgabedateien des Analysewerkzeugs verwendet. Die Dateien wurden in Excel übertragen, um die Korrelationen mit der Benchmark zu berechnen. Eine Anpassung für Web-Frontend oder Analyse war nicht erforderlich.

5. Analyse und Evaluation

In Kapitel 3 wurde die Methodik der drei Analysen vorgestellt, die im Kontext der vorliegenden Arbeit durchgeführt wurden. Dieses Kapitel präsentiert die Ergebnisse der Analysen und stellt die abgeleiteten Handlungsempfehlungen für zukünftige Projekte dar.
Um Anonymität für die Betreuer und Teilnehmer der untersuchten Projekte zu gewährleisten, werden die Projekte in diesem Kapitel mit Buchstaben von A bis F bezeichnet. Ein Projekt entspricht immer dem gleichen Buchstaben und auch die zugeordnete Farbe ist über das ganze Kapitel hinweg dieselbe.

5.1. Ergebnisse der Qualitätsanalyse

Ziel der Qualitätsanalyse war es, eine vergleichende Benchmark aller untersuchten Projekte zu erstellen. Die Projekte wurden dazu nach der in Abschnitt 3.1 vorgestellten Methodik nach den Teilmerkmalen Modularisierung, Wiederverwendbarkeit, Analysierbarkeit, Modifizierbarkeit und Testbarkeit analysiert. Die Benchmark ergibt sich aus der Summe der Bewertungen der Teilmerkmale.

Projekt A erreichte in drei der fünf Teilmerkmale die beste Bewertung. Insgesamt hat Projekt A mit einer Qualitätsnote von 4,905 auch die beste Gesamtbewertung. Die Projekte B und C erreichten jeweils in einem Teilmerkmal die beste Bewertung und belegten damit Platz zwei und drei in der Gesamtwertung. Projekt F erhielt in drei der fünf Teilmerkmale die schlechteste Bewertung und schnitt damit in der Gesamtbewertung ebenfalls am schlechtesten ab. Die Projekte können daher in eher gute und eher schlechte Projekte eingeordnet werden. Das Abschneiden der Projekte scheint also nicht nur vom Zufall bestimmt zu sein (siehe Tabelle 5.1 und Abbildung 5.1).
In vier der fünf Teilmerkmale schnitt das schlechteste Projekt mit einer Bewertung kleiner 0,7 ab; in einem Fall sogar nur knapp über 0,5. Das beste Projekt schneidet demnach in allen Teilmerkmalen mindestens 1,4 mal besser ab als das Projekt mit der schlechtesten Bewertung. Auch in der Gesamtbewertung beträgt der Faktor zwischen dem schlechtesten und besten Projekt in etwa 1,4 (siehe Abbildungen 5.1 und 5.2).

Projekt	Bench-mark	Modulari-sierung	Wieder-verwend-barkeit	Analysier-barkeit	Modifizier-barkeit	Testbar-keit
Projekt A	4,905	0,978	0,928	1,000	1,000	1,000
Projekt B	4,475	1,000	0,960	0,534	0,998	0,984
Projekt C	4,069	0,738	1,000	0,729	0,777	0,825
Projekt D	4,030	0,696	0,752	0,768	0,916	0,899
Projekt E	3,916	0,797	0,745	0,714	0,901	0,758
Projekt F	3,389	0,646	0,684	0,588	0,800	0,671

Tabelle 5.1.: Tabellarische Darstellung der Benchmark

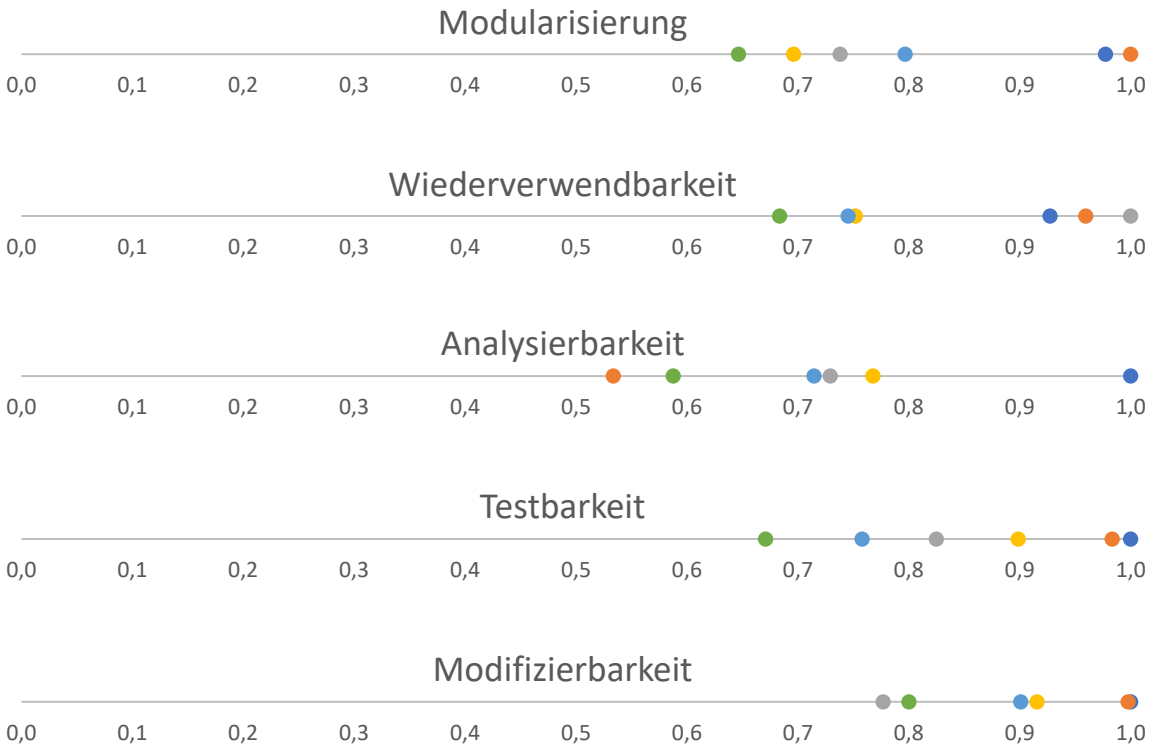

Abbildung 5.1.: Benchmark-Ergebnisse der Teilmerkmale

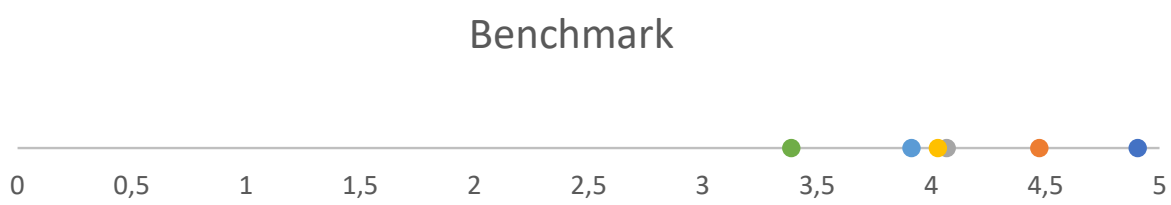

Abbildung 5.2.: Aufsummierte Benchmark aller Teilmerkmale

5.2. Ergebnisse der Repository-Analyse

In diesem Abschnitt werden die Ergebnisse der Repository-Analyse, die in Abschnitt 3.3 beschrieben wurde, dargelegt. Die Ziele der Analyse waren die Erfassung von Faktoren, die die Benchmark der Qualität beeinflussen, und die Validierung der Methodik der Qualitätsanalyse.

Commits nach Kalendertagen

Ob die Studierenden kontinuierlich oder kurz vor den Abgaben sehr viel am Projekt arbeiten, schlägt sich auch in der Qualität der entstehenden Software nieder. Der Variationskoeffizient der Verteilung der Commits nach Kalendertagen weist mit einem Korrelationskoeffizienten von r=-0,506882237 eine mittlere lineare Korrelation mit der Benchmark auf (siehe Tabelle 5.2). Arbeiten die Studierenden hauptsächlich kurz vor den Abgaben und unter Zeitdruck, so scheinen darunter die Qualität und die Wartbarkeit zu leiden.

Projekt	Mittelwert	Standardabweichung	Variationskoeffizient
Projekt A	3,420	4,727	1,382
Projekt B	1,609	3,050	1,896
Projekt C	4,477	9,595	2,143
Projekt D	12,085	17,139	1,418
Projekt E	2,480	6,610	2,666
Projekt F	3,175	6,440	2,028

Tabelle 5.2.: Tabellarische Darstellung der Commits nach Kalendertagen

In Abbildung 5.3 und Abbildung 5.4 sind die Commits von Projekt A und Projekt F tageweise in ein Balkendiagramm aufgetragen. Bei Projekt F wechseln sich Tage mit sehr vielen Commits ab mit Tagen, an denen gar nicht bzw. fast gar nicht gearbeitet wurde. Bei Projekt A sind die Unterschiede erkennbar geringer.

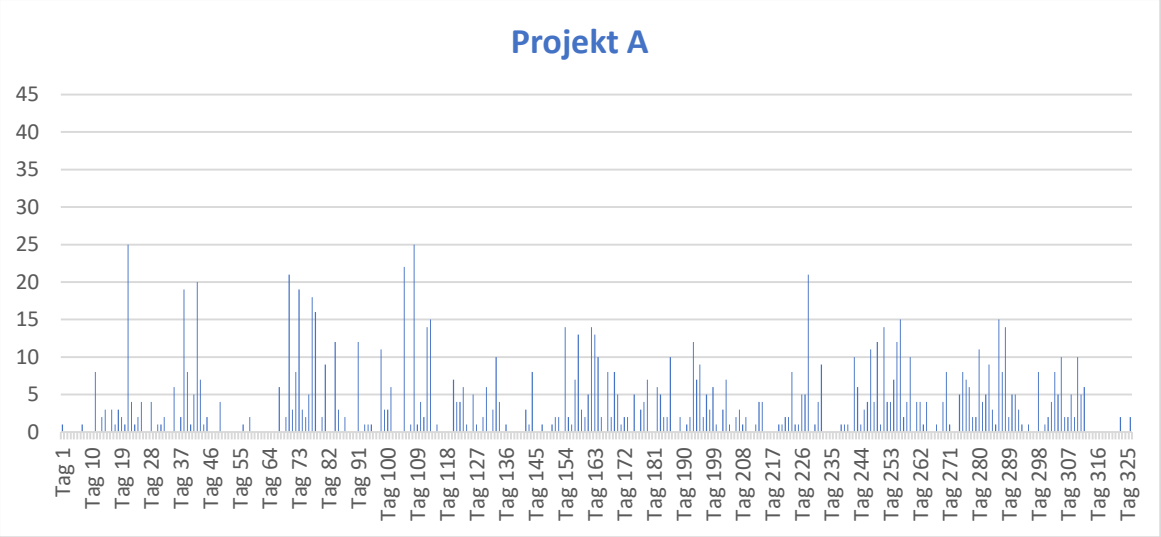

Abbildung 5.3.: Verteilung der Commits von Projekt A

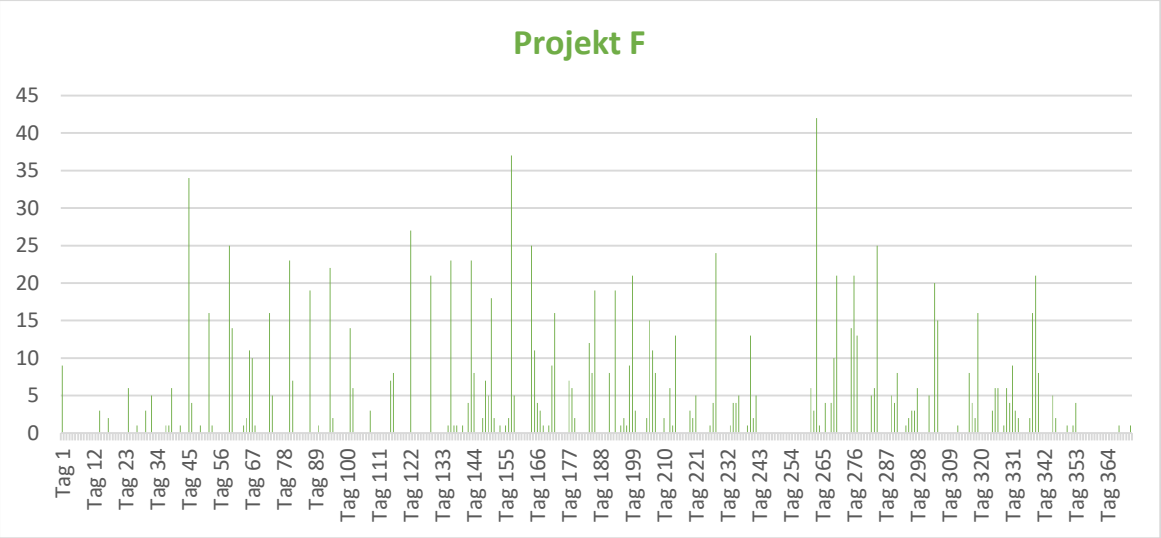

Abbildung 5.4.: Verteilung der Commits von Projekt F

Commits nach Wochentagen

Wie viele Commits an welchem Wochentag erzeugt wurden, weist eine starke lineare Korrelation mit der Benchmark auf. Sowohl der Variationskoeffizient dieser Verteilung als auch der Anteil der am Wochenende erzeugten Commits weisen einen hohen Korrelationskoeffizienten auf (r=-0,797959047 bzw. r=0,787112788). Es ist für die Qualität der Projekte von Vorteil, wenn die Teilnehmer auch außerhalb der wöchentlichen Arbeitstreffen und am Wochenende am Projekt arbeiten. Möglicherweise können sie sich zu Hause besser konzentrieren (siehe Tabellen 5.3 und 5.4).

Projekt	Mittelwert	Standardabweichung	Variationskoeffizient
Projekt A	159,286	74,674	0,469
Projekt B	89,857	51,745	0,576
Projekt C	492,429	334,782	0,680
Projekt D	630,143	524,474	0,832
Projekt E	263,571	133,200	0,505
Projekt F	168,714	197,239	1,169

Tabelle 5.3.: Tabellarische Darstellung der Commits nach Wochentagen

Projekt	Commits an Werktagen	Commits am Wochenende	Anteil der Commits am Wochenende
Projekt A	973	142	12,7%
Projekt B	549	80	12,7%
Projekt C	3012	435	12,6%
Projekt D	3852	559	12,7%
Projekt E	1680	165	8,9%
Projekt F	1161	20	1,7%

Tabelle 5.4.: Tabellarische Darstellung der Commits am Wochenende

In Abbildung 5.5 und Abbildung 5.6 kann die Anzahl der Commits für jeden Wochentag von Projekt A und Projekt F abgelesen werden. Während die Teilnehmer von Projekt F an manchen Tagen nahezu überhaupt nicht gearbeitet haben, sind die Schwankungen bei Projekt A deutlich geringer. Doch auch hier kann man erkennen, dass sich das Team vermutlich montags getroffen hat, um gemeinsam am Projekt zu arbeiten.

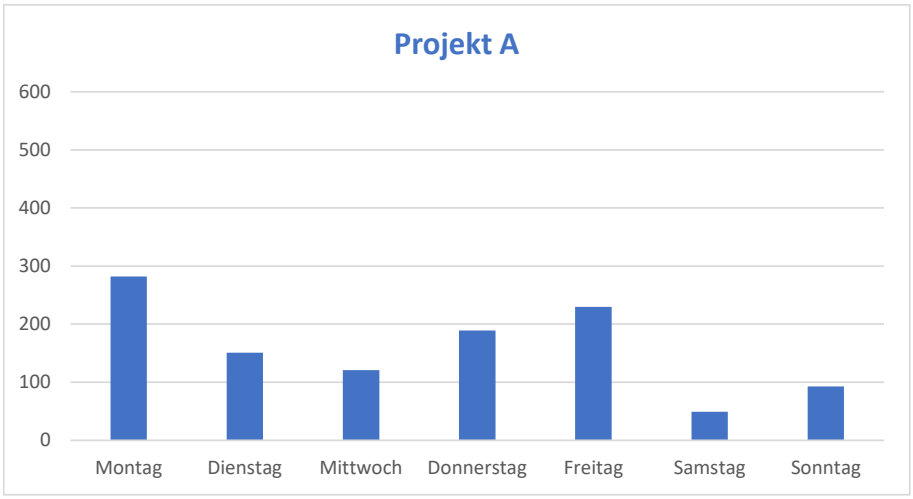

Abbildung 5.5.: Commits nach Wochentagen von Projekt A

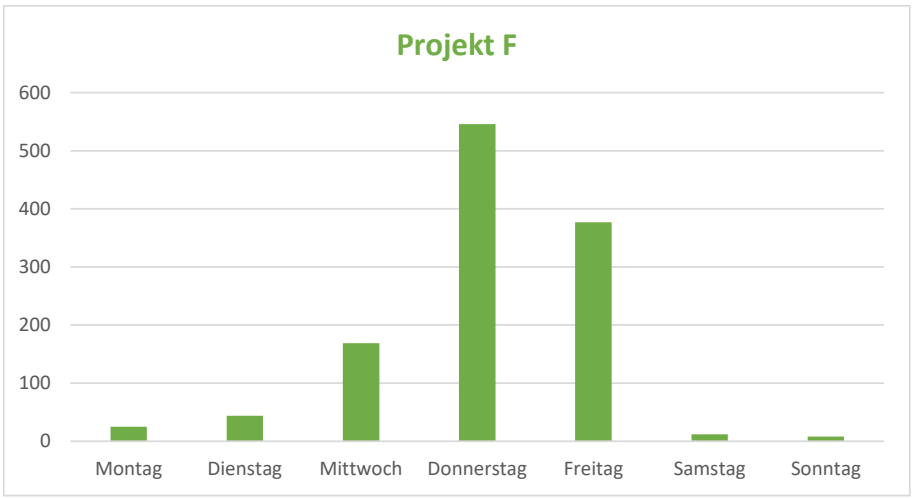

Abbildung 5.6.: Commits nach Wochentagen von Projekt F

Anzahl logischer Codezeilen

Die Anzahl der logischen Codezeilen weist eine mittlere lineare Korrelation (r=-0,636011674) mit der Benchmark auf (siehe Tabelle 5.5). Der Korrelationskoeffizient ist negativ, die untersuchten Projekte schnitten also besser ab, je geringer die Anzahl der logischen Codezeilen war. Das ist ein zu erwartendes Ergebnis, da die Qualität von großen Systemen häufig durch Software-Erosion abnimmt. Eine mögliche Verzerrung dieser Analyse könnte die Beschränkung auf Java-Code darstellen. Da während der Qualitätsanalyse nur Code untersucht wurde, der in Java geschrieben ist, umfassen die gemessenen Codezeilen auch nur den Code in Java-Klassen.

Projekt	Anzahl logischer Codezeilen
Projekt A	13969
Projekt B	10267
Projekt C	27020
Projekt D	24031
Projekt E	26756
Projekt F	21489

Tabelle 5.5.: Tabellarische Darstellung der logischen Codezeilen

Weitere Ergebnisse

Keine Korrelation mit der Benchmark konnte für die untersuchten Aspekte Verteilung der geänderten Zeilen pro Tag, Verteilung der Commits pro Teilnehmer, Verteilung der geänderten Zeilen pro Teilnehmer und die Verteilung der geänderten Zeilen pro Commit festgestellt werden (Daten siehe Anhang A.3). Diese scheinen die Benchmark nicht spürbar zu beeinflussen und es müssen daher keine Maßnahmen in diesen Bereichen ergriffen werden.

5.3. Ergebnisse der Betreuungsanalyse

Der Fragebogen zur Betreuungsanalyse wurde 11 mal ausgefüllt und zurückgeschickt. Insgesamt haben sechs Betreuer ihre Meinung abgegeben, einige von ihnen für mehrere Projekte. Zu zwei Projekten gab es je drei Rückmeldungen, ein Projekt wurde von zwei Betreuern evaluiert und bei drei Projekten hat je ein Betreuer den Fragebogen ausgefüllt und zurückgeschickt.

Wenn es mehrere Rückmeldungen zu einem Projekt gab, unterschieden sich diese teilweise deutlich in den Einschätzungen der Komplexität des fachlichen Hintergrunds und der technischen Umsetzung, der Qualität, der Betreuungsqualität, der technischen Kenntnisse der Betreuer und der Verteilung der Leistungen unter den Studierenden. Außerdem gab es bei einem Projekt Abweichungen in den Fragen, ob Refactoring-Sprints durchgeführt wurden und ob sich die Studierenden regelmäßig trafen, um zusammen am Projekt zu arbeiten. Erklären kann man dies möglicherweise mit der langen Zeit zwischen Durchführung der Projekte und Ausfüllen des Fragebogens. Außerdem können auch subjektive Einschätzungen oder der Vergleich mit anderen Projekten Ursachen für diese Abweichungen sein.

Teilnehmerzahl

Die Teilnehmerzahlen der untersuchten Projekte liegen zwischen sechs und 15 Teilnehmern. Der Median liegt bei 10,5 Teilnehmern und der Durchschnitt bei 10,66 Teilnehmern. Da aus den Teilnehmerzahlen ein Rückschluss auf die Projekte möglich ist und daher keine Anonymität von Betreuern und Teilnehmern gewährleistet werden kann, erfolgt an dieser Stelle keine Zuordnung der Teilnehmerzahlen.
Gleichwohl korreliert die Teilnehmerzahl stark linear (r=-0,866856596) mit der errechneten Benchmark. Eine besonders starke Korrelation kann für das Teilmerkmal *Modularisierung* (r=-0,931723232) nachgewiesen werden. Eine geringe Anzahl an Teilnehmern führt zu einer besseren Qualität der entstehenden Software. Projekte mit vielen Teilnehmern haben möglicherweise Probleme mit der Kommunikation und Koordination. Wenn Schnittstellen nicht abgesprochen werden und sich häufig ändern, dann führt das zu verstärkter Erosion der Software und damit zu einer abnehmenden Qualität und Wartbarkeit.

Qualifikation der Betreuer

Die Qualifikation der Betreuer von Studienprojekten zeichnet sich durch die fachlichen und technischen Kenntnissen aus. Die fachlichen Kenntnisse der Betreuer bei Studienprojekten beziehen sich auf Kenntnisse im Zusammenhang mit dem Thema eines Studienprojekts und den Geschäftsprozessen, welche die Software unterstützen soll. Die technischen Kenntnisse zeigen sich in Kenntnissen der verwendeten Technologien, Bibliotheken und Software-Architektur. Mit einem Korrelationskoeffizienten von r=0,787596217 bzw. r=0,748935423 kann eine stark lineare Korrelation zwischen den fachlichen und technischen Kenntnissen der Betreuer und der Benchmark beobachtet werden. Darüber hinaus korrelieren die technischen Kenntnisse der Betreuer sehr stark mit dem Teilmerkmal *Wiederverwendbarkeit* (r=0,968599707). Diese Erkenntnisse legen nahe, dass die Betreuungsqualität bei diesen Projekten hoch zu sein scheint, da die Studierenden von der Qualifikation der Betreuer profitieren. Daher verwundert es auch nicht, dass die Betreuungsqualität mit einem Korrelationskoeffizienten von r=0,640029258 ebenfalls einen mittleren linearen Zusammenhang mit der Benchmark aufweist.

Projekt	Fachliche Kenntnisse	Technische Kenntnisse	Betreuungsqualität
Projekt A	4	4	4
Projekt B	4	4	4
Projekt C	4	4	3
Projekt D	4	2	3
Projekt E	3	2	4
Projekt F	2	2	3

Tabelle 5.6.: Tabellarische Darstellung der Selbsteinschätzung der Betreuungsqualität (auf einer Skala von 1 bis 4)

Verwendung von Bibliotheken

Mit einem Korrelationskoeffizienten von r=0,789104736 schnitten solche Projekte besser ab, die viele Drittanbieter-Bibliotheken verwendeten. Besonders hoch war die Korrelation mit dem Teilmerkmal *Testbarkeit* (r=0,904201062). Denkbar wäre, dass es zu verbreiteten Bibliotheken hochwertige Referenzimplementierungen gibt, an denen sich die Teilnehmer bei der Verwendung der Bibliotheken orientieren konnten. Auch geben einige Bibliotheken Architekturentscheidungen vor, die sich positiv auf die Qualität und die Wartbarkeit der Projekte ausgewirkt haben könnten.

Projekt	Grad der Verwendung von Drittanbieter-Bibliotheken
Projekt A	4
Projekt B	4
Projekt C	2,5
Projekt D	4
Projekt E	3
Projekt F	2

Tabelle 5.7.: Tabellarische Darstellung zur Verwendung von Drittanbieter-Bibliotheken (auf einer Skala von 1 bis 4)

Validitätsprüfungen

Neben der Identifizierung von Faktoren, die die Benchmark beeinflussen, war ein weiteres Ziel der Betreuungsanalyse die Validierung der Qualitätsanalyse. Dazu wurden die Betreuer gebeten, ihre subjektive Einschätzung der Qualität und der Wartbarkeit des Quelltextes abzugeben. Mit einem Korrelationskoeffizienten von r=0,561089514 weist die Qualitätseinschätzung der Betreuer nur eine mittlere lineare Korrelation mit den Ergebnissen der Qualitätsanalyse auf. Die Korrelation der Wartbarkeitseinschätzung liegt mit r=0,134473309 sogar noch deutlich darunter.
Die Ursachen für die geringe Übereinstimmung mit der Benchmark müssen nicht in der Ermittlung der Benchmark liegen. Die geringe Anzahl untersuchter Projekte, eine andere Interpretation der Begriffe Wartbarkeit und Qualität durch die Betreuer oder der lange Zeitraum zwischen Abschluss der Projekte und Durchführung der Umfrage können Gründe für diese Abweichung sein. Die ältesten Projekte wurden bereits im Jahr 2010 durchgeführt, daher liegt es nahe, dass die Betreuer sich nicht mehr an Details der Projekte erinnern, die eventuell Einfluss auf die Bewertung gehabt haben könnten. Dafür spricht auch, dass die Einschätzung der Betreuer zur Arbeitsverteilung der Teilnehmer nur eine mittlere Korrelation (r=0,380005319) mit der Verteilung der Commits auf die Teilnehmer aufweist (siehe Tabelle 5.8).

Projekt	Variationskoeffizient der Commits pro Teilnehmer	Einschätzung der Heterogenität der Leistungen
Projekt A	0,774	2
Projekt B	1,119	4
Projekt C	1,151	3
Projekt D	0,806	4
Projekt E	0,690	2
Projekt F	0,793	4

Tabelle 5.8.: Vergleich der realen Verteilung der Commits auf die Studierenden und der Einschätzung der Betreuer zur Arbeitsverteilung (auf einer Skala von 1 bis 4)

Weitere Ergebnisse

Keinen Einfluss auf die Qualität der untersuchten Projekte hatten die fachliche und technische Komplexität der Projekte, der Grad der Zusammenarbeit der Studierenden, die Heterogenität der Leistungen der Teilnehmer und das Ausmaß an neuen Technologien, die zur Umsetzung verwendet wurden. Auch der von den Betreuern eingeschätzte Erfolg der Projekte korrelierte nicht mit der Benchmark (Daten siehe Anhang A.3). Vermutlich spielen in der Bewertung der Studienprojekte viele andere Aspekte eine Rolle und verringern damit den Einfluss der

Wartbarkeit. In den Aspekten Vorgehensmodell, Refactoring-Sprints, regelmäßige Arbeitstreffen und ob die Betreuer am Projekt weiterforschten und -entwickelten unterschieden sich die Projekte nicht oder nur minimal, sodass kein Einfluss auf die Benchmark berechnet werden konnte. In diesen Punkten gibt es daher keinen Anlass zur Veränderung und die Betreuer können diese Bereiche gestalten, wie sie es für richtig halten.

5.4. Handlungsempfehlungen

Kernziel der vorliegenden Arbeit war es, aus den Erkenntnissen dieser Arbeit Empfehlungen für die zukünftige Betreuung und Durchführung von Studienprojekten am Fraunhofer IAO abzuleiten. Nachfolgend werden vier Empfehlungen vorgestellt, die neue Ansätze zur Betreuung und Durchführung liefern.

Gleichmäßige Arbeitsverteilung anstreben

Die Repository-Analyse ergab, dass die Ergebnisse der Projekte besser waren, wenn sich die Commits gleichmäßig auf alle Wochentage verteilten. Die Betreuer sollten die Teilnehmer daher motivieren, auch außerhalb der vorgeschriebenen Projekttreffen am Projekt zu arbeiten. Auch die gleichmäßige Verteilung der Commits nach Kalendertagen führt zu besseren Benchmark-Ergebnissen. Den Teilnehmern sollte daher klargemacht werden, dass die Ergebnisse ihres Projekts – und damit auch ihre Note – besser ausfallen, wenn sie mit der Bearbeitung ihrer Aufgaben nicht erst kurz vor der Abgabe beginnen.
Darüber hinaus können die Betreuer die Zeiträume zwischen den Review-Terminen verkürzen und die Studierenden damit zwingen, regelmäßig am Projekt zu arbeiten. Die Freude am Projekt wird man bei den Studierenden damit allerdings nicht unbedingt steigern. Eine andere Idee wäre es daher eine intrinsische Motivation zu schaffen, sodass die Studierenden von ihrem Projekt profitieren und nicht nur an einer guten Note interessiert sind.

Geringe Teilnehmerzahl anstreben

Nach der Analyse zur Teilnehmerzahl der untersuchten Projekte erreichen Projekte mit weniger Teilnehmern einen besseren Platz in der Benchmark. Ziel des Fraunhofer IAO sollte es daher sein, neue Projekte mit einem möglichst kleinen Team durchzuführen. Da die Teilnehmer aber gleichmäßig auf alle angebotenen Projekte verteilt werden, ist nur eine geringe Einflussnahme möglich. Möglicherweise wäre es daher eine Option, große Teams in kleine Untergruppen zu unterteilen, die möglichst autark an einem Teil des Systems arbeiten. Die Schnittstellen sollten dann von den Betreuern geprüft werden, um Probleme zu vermeiden.

Hohe Qualifikation der Betreuer beibehalten

Die Ergebnisse der Betreuungsanalyse haben gezeigt, dass Projekte in der Benchmark besser abschneiden, wenn die Betreuer eine hohe fachliche und technische Qualifikation aufweisen. Aus eigener Erfahrung ist die Qualifikation der Betreuer der Studienprojekte des Fraunhofer IAO bereits sehr hoch. Eine Beibehaltung der hohen Qualifikation ist daher erstrebenswert. Eine mögliche Maßnahme wäre, vornehmlich solche Projekte anzubieten, deren Ergebnisse die Betreuer bei ihrer Arbeit unterstützen. Damit kann die Qualifikation der Betreuer sichergestellt werden. Hinzu kommt, dass diese Maßnahme auch einen positiven Einfluss auf die Motivation der Betreuer ausübt.

Verwendung von Bibliotheken fördern

Die Betreuungsanalyse ergab, dass Projekte, die viele Drittanbieter-Bibliotheken verwenden einen besseren Platz in der Benchmark erreichen. Die Betreuer sollten daher die Studierenden ermutigen, vorhandene Funktionen aus Bibliotheken zu verwenden und sich dabei an Beispielimplementierungen zu orientieren. Durch Updates der verwendeten Bibliotheken können die Betreuer im Vergleich zur Eigenimplementierung durch die Studierenden zusätzlich ihren Wartungsaufwand reduzieren.

6. Zusammenfassung und Ausblick

In dieser Arbeit wurde die Qualität von sechs Studienprojekten untersucht, die in den letzten Jahren am Fraunhofer IAO in Kooperation mit der Universität Stuttgart durchgeführt wurden. Darüber hinaus wurden zwei weitere Analysen durchgeführt, die mögliche Einflussfaktoren auf die Qualität der Projekte ermitteln sollten.
Die Qualitätsanalyse dieser Arbeit ist in etwa vergleichbar mit Qualitätsanalysen im kommerziellen Projektumfeld. Auch bei diesen Analysen kommen die in Abschnitt 2.2 vorgestellten Methoden zum Einsatz und auch Automatisierung ist dort von hoher Bedeutung, da die Analysen meist zur Überwachung der Qualität von Projekten eingesetzt werden. Die Zielsetzung ist jedoch eine andere: während es bei der Qualitätsanalyse der vorliegenden Arbeit um eine Einschätzung der Gesamtqualität der untersuchten Software ging, ist es im industriellen Umfeld von großer Bedeutung, von einer ungenügenden Gesamtqualität auf die mangelhaften Komponenten schließen zu können, um Korrekturen vorzunehmen und Verbesserung der Qualität zu erreichen.
Zur Analyse der Qualität wurde eine Benchmark entwickelt, auf der die untersuchten Projekte hinsichtlich ihrer Wartbarkeit eingeordnet wurden. Diese Benchmark legte offen, dass die Projekte sich teilweise deutlich in ihrer Qualität unterscheiden. Zwischen dem schlechtesten und dem besten Projekt lag der Faktor 1,4. Bei der Analyse der Teilmerkmale bot sich ein ähnliches Bild. Teilweise war hier die Streuung der Projekte sogar noch größer.
Die Analyse der Quellcode-Repositories der untersuchten Projekte ergab, dass die Verteilung der Arbeit nach Wochentagen und über den gesamten Projektverlauf die Qualität der entstehenden Software maßgeblich beeinflusst. Daraus leitet sich die Empfehlung ab, die Teilnehmer zu animieren, sich ihre Arbeit gleichmäßig einzuteilen und nicht nur einen Tag in der Woche oder kurz vor den Abgaben am Projekt zu arbeiten.
Die Betreuungsanalyse zeigte auf, dass die Teilnehmerzahl, die Qualifikation der Betreuer und die Verwendung von Drittanbieter-Bibliotheken ebenfalls deutlichen Einfluss auf die Qualität des Quellcodes nehmen. Auch hier wurden Empfehlungen zur Verbesserung der Betreuung und Durchführung von zukünftigen Studienprojekten am Fraunhofer IAO abgeleitet.

Ausblick

Die Analysen dieser Arbeit können auf zukünftige Studienprojekte am Fraunhofer IAO oder anderen Instituten angewendet werden. Durch die Verwendung der Benchmark kann so sehr einfach festgestellt werden, ob sich die Qualität der Projekte verändert. Auch die Repository-Analyse kann größtenteils automatisiert angewendet werden. Einzig die Betreuungsanalyse würde einen etwas höheren Aufwand erfordern, da diese manuell durchzuführen ist.

Anknüpfend an diese Arbeit könnten die automatisierten Analysen auch auf das Softwarepraktikum angewendet werden, das ebenfalls an der Universität Stuttgart durchgeführt wird. Beim Softwarepraktikum arbeitet ein ganzer Jahrgang des Studiengangs Softwaretechnik in Dreier-Teams über ein Semester lang an der gleichen Aufgabenstellung. Die Ergebnisse der Teams haben daher eine viel höhere Vergleichbarkeit, als die in dieser Arbeit untersuchten Studienprojekte. Außerdem hätte man so direkt eine viel größere Anzahl an Datenpunkten, da das Softwarepraktikum typischerweise jedes Jahr mit mehr als 20 Gruppen durchgeführt wird. Aufgrund der höheren Vergleichbarkeit könnte eine Analyse der Projekte auch während des Projekts zu bestimmten Meilensteinen durchgeführt werden, um einen Eindruck vom Verlauf der Projekte zu bekommen.

A. Anhang

A.1. Installation des Werkzeugs

Im Folgenden wird die Installation des entstandenen Werkzeugs beschrieben. Dabei wird davon ausgegangen, dass ein Windows-System verwendet wird. Grundsätzlich sollte die Verwendung des Werkzeugs auch auf Linux funktionieren.

Installation von Python

Die aktuelle Version von Python kann unter https://www.python.org/downloads/ heruntergeladen werden. Installieren Sie Python mithilfe des Installers.

Installation von Git

Installieren Sie die aktuelle Version von Git unter https://git-scm.com/download. Es sind keine besonderen Konfigurationen notwendig.

Installation der Python-Bibliotheken

Öffnen Sie eine Eingabeaufforderung und führen Sie nacheinander die Befehle *pip install gitpython*, *pip install pyyaml*, *pip install numpy* und *pip install python-dateutil* aus.

Installation von Node.js und npm

Die aktuelle Version von Node.js und npm finden Sie unter https://nodejs.org/en/download/. Laden Sie den Installer herunter und folgen Sie den Anweisungen zur Installation.

Installation von http-server

Öffnen Sie eine Eingabeaufforderung und führen Sie den Befehl *npm install http-server -g* aus, um den http-server als globales Modul auf ihrem System zu installieren.

Installation von SourceMeter

Unter https://www.sourcemeter.com/download/ können Sie die aktuelle Version von SourceMeter beantragen. Sie erhalten den Download-Link an die von Ihnen angegebene E-Mail-Adresse. Laden Sie das Archiv herunter, entpacken Sie es und legen Sie die Dateien in einem Verzeichnis ihrer Wahl ab.

Installation von Java

Laden Sie unter https://java.com/de/download/ die aktuelle Java-Version herunter und installieren Sie diese mittels des Installers.

Konfiguration des Werkzeugs

Das Werkzeug kann über eine *yaml*-Konfigurationsdatei konfiguriert werden. In diese Datei trägt der Nutzer folgende Daten ein:

- Die Projekte mit ihrem Verzeichnis, ihrem Namen und dem Versionsstand auf dem die Analyse durchgeführt werden soll
- Den Pfad zur *SourceMeter*-Anwendung und eine Filterdatei, um einzelne Verzeichnisse aus der Analyse auszuschließen
- Den Pfad zum Quellcode des Web-Frontends, um für jedes Projekt ein eigenes Web-Frontend erstellen zu können
- Ein Ausgabeverzeichnis, in das die Ergebnisdateien abgelegt werden sollen

Verwendung des Werkzeugs

Öffnen Sie die Datei *config.yaml* und konfigurieren Sie das Werkzeug. Tragen Sie hierzu die zu untersuchenden Projekte, das Ausgabeverzeichnis, den Pfad zur SourceMeter-Installation und den Pfad zum *dist*-Verzeichnis des Web-Frontends ein.

Führen Sie die Datei *calculate_benchmark.py* aus und warten Sie bis die Analyse abgeschlossen ist. In dem von Ihnen ausgewählten Ausgabeverzeichnis wurden nun für jedes Projekt die Ergebnisdateien und das fertige Web-Frontend angelegt.

Um das Web-Frontend auszuführen, öffnen Sie eine Eingabeaufforderung, navigieren Sie in das Verzeichnis des Web-Frontends und führen Sie den Befehl *http-server* aus. Nun können Sie das Web-Frontend im Browser unter http://localhost:8080 erreichen.

A.2. Fragebogen zur Betreuungsanalyse

Zur Durchführung der Betreuungsanalyse aus Abschnitt 3.3 wurde den Betreuern ein Fragebogen per E-Mail zugeschickt. Nachfolgend ist dieser Fragebogen abgebildet.

Fragebogen zur Ermittlung der Qualität und Betreuung von Studienprojekten

Im Rahmen meiner Bachelorarbeit führe ich eine Untersuchung der Qualität von Studienprojekten anhand der Quellcode-Repositories durch, die in den letzten Jahren am Fraunhofer IAO durchgeführt wurden. Um ein wenig Kontext für die Projekte zu haben, würde ich Sie als Betreuer bitten, mir bei meiner Studie zu helfen und den nachfolgenden Fragebogen auszufüllen und per Mail an mich zurückzuschicken.

Falls Sie mehrere Projekte betreut haben, füllen Sie bitte für jedes Projekt eine separate Kopie dieses Fragebogens aus. Die Ergebnisse werden selbstverständlich vertraulich behandelt.

Falls Sie Fragen oder Anmerkungen zum Fragebogen haben, können Sie sich gerne per Mail an mich oder meinen Betreuer wenden.

Vielen Dank für Ihre Mithilfe,

Benedikt Kersjes

Allgemeine Fragen

Welches Projekt haben Sie betreut?

Klicken oder tippen Sie hier, um Text einzugeben.

Wie kompliziert war der fachliche Hintergrund aus Ihrer Sicht?

Sehr einfach Sehr kompliziert

☐ ☐ ☐ ☐

Wie kompliziert war die technische Umsetzung aus Ihrer Sicht?

Sehr einfach Sehr kompliziert

☐ ☐ ☐ ☐

In welchem Ausmaß wurde der erwartete Funktionsumfang umgesetzt?

Kaum umgesetzt Nahezu vollständig umgesetzt

☐ ☐ ☐ ☐

Wie schätzen Sie subjektiv den Erfolg des Projekts ein?

Kaum erfolgreich Sehr erfolgreich

☐ ☐ ☐ ☐

Fragen zur Qualität

Wie hoch schätzen Sie die Qualität des Quellcodes insgesamt ein?

Sehr gering Sehr hoch

☐ ☐ ☐ ☐

Wie hoch schätzen Sie die Wartbarkeit des Quellcodes insgesamt ein?

Sehr gering Sehr hoch

☐ ☐ ☐ ☐

Haben Sie das Projekt als Grundlage für Ihre weitere Forschung verwendet?

☐ Ja ☐ Nein ☐ Weiß ich nicht

Haben Sie am Quellcode des Projekts weiterentwickelt?

☐ Ja ☐ Nein ☐ Weiß ich nicht

Fragen zur Betreuung

Wie hoch schätzen Sie die Betreuungsqualität ein?

Sehr gering Sehr hoch

☐ ☐ ☐ ☐

Haben Sie sich die Verantwortlichkeiten untereinander aufgeteilt (z.B. fachlicher/technischer/organisatorischer Ansprechpartner)?

☐ Ja, wir haben uns aufgeteilt

☐ Nein, jeder war Ansprechpartner für alles

☐ Weiß ich nicht

Wie hoch war die Kenntnis der Betreuer über die fachlichen Themen des Studienprojekts?

Sehr gering Sehr hoch

☐ ☐ ☐ ☐

Wie hoch war die Kenntnis der Betreuer über die technischen Themen des Studienprojekts?

Sehr gering Sehr hoch

☐ ☐ ☐ ☐

Fragen zur Umsetzung

Welches Vorgehensmodell haben die Studierenden verwendet?

Klicken oder tippen Sie hier, um Text einzugeben.

War das Vorgehensmodell vorgegeben?

☐ Ja ☐ Nein ☐ Weiß ich nicht

Falls das Vorgehensmodell Scrum war: Wurden Refactoring-Sprints durchgeführt?

☐ Ja ☐ Nein ☐ Weiß ich nicht

Haben sich die Studierenden regelmäßig getroffen, um am Projekt zu arbeiten?

☐ Ja ☐ Nein ☐ Weiß ich nicht

Wie gut war die Zusammenarbeit der Studierenden aus Ihrer Sicht?

Sehr schlecht Sehr gut

☐ ☐ ☐ ☐

Haben die Studierenden eher gleiche Leistungen erbracht oder waren die Leistungen der Studierenden heterogen?

Eher gleich Eher ungleich

☐ ☐ ☐ ☐

Wurden eher wenige oder eher viele Drittanbieter-Bibliotheken verwendet?

Eher wenige Eher viele

☐ ☐ ☐ ☐

Wurden bewährte Technologien verwendet oder neue Technologien erprobt?

Viele bewährte Technologien Viele neue Technologien

☐ ☐ ☐ ☐

A.3. Weitere Daten der Analysen

In diesem Kapitel sind die ermittelten Daten der Repository-Analyse und der Betreuungsanalyse aufgeführt, für die kein Zusammenhang mit der Benchmark ermittelt werden konnte.

Projekt	Mittelwert	Standardabweichung	Variationskoeffizient
Projekt A	1272,310	3857,993	3,032
Projekt B	8752,997	94241,504	10,767
Projekt C	4378,547	35212,817	8,042
Projekt D	2274,838	6829,125	3,002
Projekt E	1899,629	8261,468	4,349
Projekt F	1399,790	8524,505	6,090

Tabelle A.1.: Tabellarische Darstellung der geänderten Zeilen pro Tag

Projekt	Mittelwert	Standardabweichung	Variationskoeffizient
Projekt A	159,286	123,351	0,774
Projekt B	78,625	87,977	1,119
Projekt C	172,350	198,368	1,151
Projekt D	245,056	197,624	0,806
Projekt E	131,786	90,985	0,690
Projekt F	73,813	58,511	0,793

Tabelle A.2.: Tabellarische Darstellung der Commits pro Teilnehmer

Projekt	Mittelwert	Standardabweichung	Variationskoeffizient
Projekt A	51761,857	38697,254	0,748
Projekt B	427802,750	1010418,701	2,362
Projekt C	168574,050	252871,137	1,500
Projekt D	46128,667	39319,690	0,852
Projekt E	100951,714	84900,076	0,841
Projekt F	32545,125	43821,516	1,346

Tabelle A.3.: Tabellarische Darstellung der geänderten Zeilen pro Teilnehmer

Projekt	Mittelwert	Standardabweichung	Variationskoeffizient
Projekt A	324,962	1500,003	4,616
Projekt B	5441,052	52128,408	9,581
Projekt C	978,091	15160,940	15,501
Projekt D	188,238	1344,150	7,141
Projekt E	766,029	3531,816	4,611
Projekt F	440,916	4540,663	10,298

Tabelle A.4.: Tabellarische Darstellung der geänderten Zeilen pro Commit

Projekt	Fachliche Komplexität	Technische Komplexität
Projekt A	3	3
Projekt B	3	2
Projekt C	4	4
Projekt D	2	4
Projekt E	2	3
Projekt F	3	3

Tabelle A.5.: Einschätzung der Komplexität durch die Betreuer (auf einer Skala von 1 bis 4)

Projekt	Zusammenarbeit
Projekt A	4
Projekt B	2
Projekt C	2
Projekt D	2
Projekt E	4
Projekt F	3

Tabelle A.6.: Einschätzung der Zusammenarbeit durch die Betreuer (auf einer Skala von 1 bis 4)

Projekt	Heterogenität der Leistungen
Projekt A	2
Projekt B	4
Projekt C	3
Projekt D	4
Projekt E	2
Projekt F	4

Tabelle A.7.: Einschätzung der Heterogenität der Leistungen durch die Betreuer (auf einer Skala von 1 bis 4)

Projekt	Menge an neuen Technologien
Projekt A	2
Projekt B	1
Projekt C	1,5
Projekt D	4
Projekt E	1
Projekt F	2

Tabelle A.8.: Einschätzung des Ausmaßes neuer Technologien durch die Betreuer (auf einer Skala von 1 bis 4)

Projekt	Erfolg
Projekt A	4
Projekt B	3
Projekt C	4
Projekt D	2
Projekt E	3
Projekt F	4

Tabelle A.9.: Einschätzung des Erfolgs durch die Betreuer (auf einer Skala von 1 bis 4)

Literaturverzeichnis

[AC94] F. B. Abreu, R. Carapuça. „Object-oriented software engineering: Measuring and controlling the development process“. In: *Proceedings of the 4th international conference on software quality*. Bd. 186. 1994, S. 1–8 (zitiert auf S. 23, 24).

[Aht03] T. Ahtee. „Inspections and historical data in teaching software engineering project course“. In: *Software Engineering Education and Training, 2003.(CSEE&T 2003). Proceedings. 16th Conference on*. IEEE. 2003, S. 288–297 (zitiert auf S. 15).

[AM96] F. B. e Abreu, W. Melo. „Evaluating the impact of object-oriented design on software quality“. In: *Software Metrics Symposium, 1996., Proceedings of the 3rd International*. IEEE. 1996, S. 90–99 (zitiert auf S. 31).

[Bas] BasLeijdekkers. *BasLeijdekkers/MetricsReloaded: Automated code metrics plugin for IntelliJ IDEA*. https://github.com/BasLeijdekkers/MetricsReloaded. [Online; abgerufen am 14.07.2017] (zitiert auf S. 27).

[BDKT00] B. Bruegge, A. H. Dutoit, R. Kobylinski, G. Teubner. „Transatlantic project courses in a university environment“. In: *Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific*. IEEE. 2000, S. 30–37 (zitiert auf S. 11).

[BDW99] L. C. Briand, J. W. Daly, J. K. Wust. „A unified framework for coupling measurement in object-oriented systems“. In: *IEEE Transactions on software Engineering* 25.1 (1999), S. 91–121 (zitiert auf S. 31).

[BHSV04] L. Bischofs, W. Hasselbring, J. Sauer, O. Vornberger. „Das Virtuelle Softwareprojekt“. In: (2004) (zitiert auf S. 11).

[BKA15] B. Bruegge, S. Krusche, L. Alperowitz. „Software engineering project courses with industrial clients“. In: *ACM Transactions on Computing Education (TOCE)* 15.4 (2015), S. 17 (zitiert auf S. 11).

[BV04] M. Bruntink, A. Van Deursen. „Predicting class testability using object-oriented metrics“. In: *Source Code Analysis and Manipulation, 2004. Fourth IEEE International Workshop on*. IEEE. 2004, S. 136–145 (zitiert auf S. 22, 31).

[CDK98] S. R. Chidamber, D. P. Darcy, C. F. Kemerer. „Managerial use of metrics for object-oriented software: An exploratory analysis“. In: *IEEE Transactions on software Engineering* 24.8 (1998), S. 629–639 (zitiert auf S. 31).

[che] checkstyle. *checkstyle - Checkstyle 8.0*. http://checkstyle.sourceforge.net/. [Online; abgerufen am 14.07.2017] (zitiert auf S. 28).

[CK94] S. R. Chidamber, C. F. Kemerer. „A metrics suite for object oriented design“. In: *IEEE Transactions on software engineering* 20.6 (1994), S. 476–493 (zitiert auf S. 23–25).

[CLC09] C. Costa-Soria, M. Llavador, M. del Carmen Penades. „An approach for teaching software engineering through reverse engineering“. In: *EAEEIE Annual Conference, 2009*. IEEE. 2009, S. 1–6 (zitiert auf S. 11).

[CMRW77] G. E. Company, J. A. McCall, P. K. Richards, G. F. Walters. *Factors in software quality*. Information Systems Programs, General Electric Company, 1977 (zitiert auf S. 17, 18).

[Con68] M. E. Conway. „How do committees invent“. In: *Datamation* 14.4 (1968), S. 28–31 (zitiert auf S. 12).

[CR94] V. Caldiera, H. D. Rombach. „The goal question metric approach“. In: *Encyclopedia of software engineering* 2.1994 (1994), S. 528–532 (zitiert auf S. 19, 20).

[DHZS99] B. Demuth, H. Hußmann, S. Zschaler, L. Schmitz. „Erfahrungen mit einem frameworkbasierten Softwarepraktikum“. In: *Software Engineering im Unterricht der Hochschulen SEUH* 99 (1999), S. 21–30 (zitiert auf S. 11).

[DJLW09] F. Deissenboeck, E. Juergens, K. Lochmann, S. Wagner. „Software quality models: Purposes, usage scenarios and requirements“. In: *Software Quality, 2009. WOSQ'09. ICSE Workshop on*. IEEE. 2009, S. 9–14 (zitiert auf S. 16).

[EHD01] L. H. Etzkorn, W. E. Hughes, C. G. Davis. „Automated reusability quality analysis of OO legacy software“. In: *Information and Software Technology* 43.5 (2001), S. 295–308 (zitiert auf S. 22, 25, 31).

[Fac] Fachbereich Informatik, Universität Stuttgart. *Studienprojekte: Informationen für Studierende | Der Fachbereich Informatik | Universität Stuttgart*. https://www.informatik.uni-stuttgart.de/studium/studierende/bsc-studiengaenge/softwaretechnik/studienprojekte/studi-infos-index.html. [Online; abgerufen am 12.07.2017] (zitiert auf S. 11).

[Fin] FindBugs. *FindBugs™- Find Bugs in Java Programs*. http://findbugs.sourceforge.net/. [Online; abgerufen am 14.07.2017] (zitiert auf S. 28).

[Fro] FrontEndART Ltd. *SourceMeter - Free-to-use, Advanced Source Code Analysis Suite*. https://www.sourcemeter.com/. [Online; abgerufen am 14.07.2017] (zitiert auf S. 26).

[Git] Git. *Git - git-svn Documentation*. https://git-scm.com/docs/git-svn. [Online; abgerufen am 13.08.2017] (zitiert auf S. 33, 38).

[GSM13] S. Gokhale, T. Smith, R. McCartney. „Teaching software maintenance with open source software: Experiences and lessons“. In: *2013 IEEE Frontiers in Education Conference (FIE)*. IEEE. 2013, S. 1664–1670 (zitiert auf S. 11).

[Ham06] T. Hampp. „Quantitative Analyse studentischer Projekte“. In: *Softwaretechnik-Trends* 26.1 (2006) (zitiert auf S. 15).

[HCN00] R. Harrison, S. Counsell, R. Nithi. „Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems“. In: *Journal of Systems and Software* 52.2 (2000), S. 173–179 (zitiert auf S. 31).

[Hof13] D. W. Hoffmann. *Software-Qualität*. Springer-Verlag, 2013 (zitiert auf S. 21).

[Ima] Imagix. *Static Analysis, Software Metrics and Test - Imagix*. https://www.imagix.com / products / static - analysis - and - metrics . html. [Online; abgerufen am 14.07.2017] (zitiert auf S. 27).

[ISO01] ISO 9126. *Software engineering – Product quality – Part 1: Quality model*. Norm ISO/IEC 9126-1:2001. Genf, CH: International Organization for Standardization, Juni 2001 (zitiert auf S. 16).

[ISO11] ISO 25010. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. Norm ISO/IEC 25010:2011. Genf, CH: International Organization for Standardization, März 2011 (zitiert auf S. 16).

[LH93] W. Li, S. Henry. „Object-oriented metrics that predict maintainability“. In: *Journal of systems and software* 23.2 (1993), S. 111–122 (zitiert auf S. 23, 31).

[LL13] J. Ludewig, H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. dpunkt. verlag, 2013 (zitiert auf S. 21).

[M S] M Squared Technologies. *Resource Standard Software Source Code Metrics For C, C++, C# and Java*. http://msquaredtechnologies.com/m2rsm/index.htm. [Online; abgerufen am 14.07.2017] (zitiert auf S. 27).

[MBD+09] K. Mordal-Manet, F. Balmas, S. Denier, S. Ducasse, H. Wertz, J. Laval, F. Bellingard, P. Vaillergues. „The squale model—A practice-based industrial quality model“. In: *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE. 2009, S. 531–534 (zitiert auf S. 18).

[McC76] T. J. McCabe. „A complexity measure“. In: *IEEE Transactions on software Engineering* 4 (1976), S. 308–320 (zitiert auf S. 24).

[Mey88] B. Meyer. *Object-oriented software construction*. Bd. 2. Prentice hall New York, 1988 (zitiert auf S. 31).

[OW14] J.-P. Ostberg, S. Wagner. „On Automatically Collectable Metrics for Software Maintainability Evaluation“. In: *Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2014 Joint Conference of the International Workshop on*. IEEE. 2014, S. 32–37 (zitiert auf S. 31).

[Pea95] K. Pearson. „Note on regression and inheritance in the case of two parents“. In: *Proceedings of the Royal Society of London* 58 (1895), S. 240–242 (zitiert auf S. 34).

[PMD] PMD. *PMD*. https://pmd.github.io/. [Online; abgerufen am 14.07.2017] (zitiert auf S. 28).

[Pow] Power Software. *Power Software - Products - Essential Metrics*. http://www.powersoftware.com/em/. [Online; abgerufen am 14.07.2017] (zitiert auf S. 27).

[PSB11] W. Poncin, A. Serebrenik, M. van den Brand. „Mining student capstone projects with FRASR and ProM“. In: *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM. 2011, S. 87–96 (zitiert auf S. 16).

[PSV11] W. Poncin, A. Serebrenik, M. Van Den Brand. „Process mining software repositories“. In: *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*. IEEE. 2011, S. 5–14 (zitiert auf S. 16).

[sau] sauerf. *Metrics 1.3.6*. http://metrics.sourceforge.net/. [Online; abgerufen am 14.07.2017] (zitiert auf S. 27).

[Sem] Semantic Designs. *Semantic Designs: Java Source Code Metrics*. http://www.semanticdesigns.com/Products/Metrics/JavaMetrics.html. [Online; abgerufen am 14.07.2017] (zitiert auf S. 27).

[Son] SonarSource S.A. *Continuous Code Quality | SonarQube*. https://www.sonarqube.org/. [Online; abgerufen am 14.07.2017] (zitiert auf S. 26).

[Vir] Virtual Machinery. *JHawk - the Java metrics tool - Product Overview*. http://www.virtualmachinery.com/jhawkprod.htm. [Online; abgerufen am 14.07.2017] (zitiert auf S. 27).

[VMV+05] B. F. Van Dongen, A. K. A. de Medeiros, H. Verbeek, A. Weijters, W. M. Van Der Aalst. „The prom framework: A new era in process mining tool support.“ In: *ICATPN*. Bd. 3536. Springer. 2005, S. 444–454 (zitiert auf S. 16).

[Wal] L. Walton. *Eclipse Metrics Plugin - State Of Flow*. http://eclipse-metrics.sourceforge.net/. [Online; abgerufen am 14.07.2017] (zitiert auf S. 27).

[WLH+12] S. Wagner, K. Lochmann, L. Heinemann, M. Kläs, A. Trendowicz, R. Plösch, A. Seidl, A. Goeb, J. Streit. „The quamoco product quality modelling and assessment approach“. In: *Proceedings of the 34th international conference on software engineering*. IEEE Press. 2012, S. 1133–1142 (zitiert auf S. 20).

[Zus93] H. Zuse. „Criteria for program comprehension derived from software complexity metrics“. In: *Program Comprehension, 1993. Proceedings., IEEE Second Workshop on*. IEEE. 1993, S. 8–16 (zitiert auf S. 31).

Alle URLs wurden zuletzt am 26. 09. 2017 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift