Existiert eine Korrelation zwischen Storypoint-Aufwandsabschätzungen und Softwarekomplexitätsmetriken?

Eine deskriptive Fallstudie sechs agiler Softwareprojekte

Bachelorarbeit

vorgelegt am 7. Mai 2022

Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik

Kurs WI2019I

von

Tim Struthoff

Betreuer in der Ausbildungsstätte:

DXC Technologies
Andreas Jordan
Managing Consultant

Unterschrift der Betreuerin/des Betreuers

DHBW Stuttgart:

Katja Sattler
Delivery Lead, Testing and Digital Assurance

Inhaltsverzeichnis

Abkürzungsverzeichnis **IV**

Abbildungsverzeichnis **V**

Tabellenverzeichnis **VI**

1 Einleitung **1**
1.1 Zielsetzung . . . 1
1.2 Verwandte Arbeiten . . . 2
1.3 Forschungsbeitrag . . . 3
1.4 Methodisches Vorgehen . . . 3
1.5 Aufbau der Arbeit . . . 5

2 Softwarekomplexitätsmaße **6**
2.1 Definitionen . . . 6
2.2 Kritik . . . 7
2.3 Eine Auswahl von Softwarekomplexitätsmaßzahlen . . . 8
2.3.1 Logische Codezeilen . . . 10
2.3.2 Zyklomatische Komplexität . . . 10
2.3.3 Halstead Metriken . . . 12
2.3.4 Einrückungskomplexität . . . 14
2.3.5 Einrückungskomplexität . . . 16

3 Aufwandsabschätzungen agiler Projekte **19**
3.1 Die agile Arbeitsweise in der Softwareentwicklung . . . 19
3.2 Methoden der agilen Softwareentwicklung . . . 19
3.2.1 Ablauf . . . 20
3.2.2 Aufwandsabschätzungen mit Planning Poker . . . 20

4 Forschungsaufbau der Fallstudie **21**
4.1 Forschungsfrage . . . 21
4.2 Hypothesen . . . 21
4.3 Analyseeinheit . . . 22
4.4 Datensammlung . . . 23
4.5 Verbindung von Daten und Hypothesen . . . 24
4.6 Kriterien zur Interpretation der Daten . . . 26
4.7 Implementierung einer Untersuchungssoftware . . . 28
4.7.1 Anforderungen an die Untersuchungssoftware . . . 28
4.7.2 Aufbau der Untersuchungssoftware . . . 30
4.7.3 Zahlenverarbeitungsmodul . . . 31
4.8 Grenzen und Probleme . . . 38

5 Untersuchung der Softwarekomplexitätsmetriken **39**
5.1 Digital NDA Application . . . 39
5.1.1 Datenerhebung . . . 39
5.1.2 Auswertung . . . 40

5.1.3 Fazit . . . 42
5.1.4 Kritik und Messfehler . . . 42
5.2 inGRID . . . 42
5.2.1 Datenerhebung . . . 43
5.2.2 Auswertung . . . 44
5.2.3 Fazit . . . 45
5.2.4 Kritik . . . 46
5.3 Alstonii . . . 46
5.3.1 Datenerhebung . . . 46
5.3.2 Auswertung . . . 47
5.3.3 Fazit . . . 48
5.3.4 Kritik . . . 48
5.4 Lacustris . . . 48
5.4.1 Datenerhebung . . . 49
5.4.2 Auswertung . . . 49
5.4.3 Kritik . . . 50
5.4.4 Fazit . . . 50
5.5 Engelmannii . . . 51
5.5.1 Datenerhebung . . . 51
5.5.2 Auswertung . . . 52
5.5.3 Fazit . . . 53
5.5.4 Kritik . . . 53
5.6 GitLab . . . 53
5.6.1 Datenerhebung . . . 54
5.6.2 Auswertung . . . 54
5.6.3 Kritik . . . 55
5.6.4 Fazit . . . 55

6 Zusammenfassende Analyse der Untersuchungsergebnisse 56

7 Conclusion 58
7.1 Kritische Evaluierung . . . 59
7.2 Ausblick . . . 60
7.2.1 Zusätzliche Validierung der Ergebnisse . . . 60
7.2.2 Umgebungsparameteranalyse . . . 60

Anhang 62

Literaturverzeichnis 72

Abkürzungsverzeichnis

DIL	Digital Innovation Lab
WI	Wirtschaftsinformatik
IEEE	Institute of Electrical and Electronics Engineers
SLOC	Anzahl der logischen Codezeilen
LOC	Anzahl der Codezeilen
CSV	Comma-separated values
PNG	Portable Network Graphik
PDF	Portable Document Format
UTC	Coordinated Universal Time
NDA	Non-Disclosure Agreement
HTML	HyperText Markup Language
VM	Virtual machine
JQL	Jira Query Language
SQL	Structured Query Language
SSH	Secure Shell
AWS	Amazon Web Services
API	Application Programming Interface

Abbildungsverzeichnis

1 Mal wieder das DHBW-Logo. 63

Tabellenverzeichnis

1 Einleitung

Diese Bachelorarbeit wird als Teil eines Praktikums in der Digital Innovation Lab (DIL) Abteilung des IT-Beratungs- und Dienstleistungsunternehmens DXC Technology geschrieben. Die DXC Technology befasst sich unter anderem mit der Herstellung von Individualsoftware für eine Vielzahl von Kunden[1]. Die DIL Abteilung im Speziellen ist dabei für die Erstellung von ersten, minimal lauffähigen Versionen (Minimum Viable Products, MVPs) dieser Softwareprodukte verantwortlich. Diese MVPs werden dann von anderen Abteilungen der DXC zu größeren Softwareprodukten weiterentwickelt.

Ein Teil des Dienstleistungsangebotes der DXC liegt in der Wartung, Betreuung und Weiterentwicklung eben dieser Software. Angebote zur Wartung und Weiterentwicklung von Software stellen auch eine wesentliche Umsatzquelle der DXC dar. Dieser Bereich ist also aus betriebswirtschaftlicher Sicht von besonderem Interesse. Auch aus der Perspektive der Kunden der DXC ist der Bereich der Wartung und Weiterentwicklung der Software von Interesse. So entfällt ein signifikanter Anteil der Kosten eines Softwareproduktes auf deren Wartung und Weiterentwicklung[2]. Im Interesse des Kunden sollten diese Kosten gesenkt werden[3].

Eine Vielzahl an Studien kamen zu dem Konsens, dass die Komplexität eines Softwareproduktes einen wesentlichen Einflussfaktor für den Aufwand von Wartung, Betrieb und Weiterentwicklung der Software darstellt. Jones 2008 konnte beweisen, dass die Komplexität und der Umfang von Software stark und direkt mit dem Wartungsaufwand[4] und der durchschnittlichen Fehleranzahl korrelieren[5]. Also sei es sinnvoll, die Komplexität der Software permanent zu beobachten[6].

1.1 Zielsetzung

Motiviert durch die betriebswirtschaftliche Relevanz der Softwarekomplexität liegt die Zielsetzung dieser Arbeit in der Auswahl und Validierung von Methoden zur automatisierten Bestimmung der Komplexität von Software. Die Auswahl der Berechnungsmethoden erfolgt anhand einer umfangreichen Literaturanalyse. Das Arbeitsergebnis ist in diesem ersten Schritt eine Aufstellung von Berechnungsmethoden mit jeweils einer Erklärung. So soll es den Leser*innen möglich sein, sich ein umfangreiches Bild der aktuellen Praxis in der Softwarekomplexitätsbestimmung zu verschaffen.

[1] Vgl. *Interview Mit Vertretern Der Abteilung* 2022
[2] Vgl. Jones 2008, S. 301
[3] Vgl. *Interview Mit Einer Mitarbeiterin Des DIL* 2022
[4] Vgl. Jones 2008, S. 64, 335 und 627
[5] Vgl. Jones 2008, S. 64 und 503
[6] Vgl. Jones 2008, S. 503

Für die Verifizierung der zuvor aufgestellten Metriken wird ein Vergleich angestellt. Dabei werden für fünf Projekte der DXC Technologies, sowie für ein externes Projekt Komplexitätsabschätzungen von Experten mit den berechneten Metriken verglichen. Als Ergebnis dieses Arbeitsschrittes ist eine Bestimmung des Grades der Korrelation vorgesehen.

Mit diesem Arbeitsergebnis soll dann in einem letzten Schritt ein Ausblick auf die weitere Verwendung der Ergebnisse gegeben werden. Insbesondere wird die zukünftige Umsetzung einer sog. Umgebungsparameteranalyse in Aussicht gestellt. Diese wurde bereits von einem unternehmensinternen Experten skizziert. Dabei soll das Ergebnis dieser Arbeit mit verschiedenen Einflussfaktoren der Projektumgebung verglichen werden. Aus diesem Vergleich sollen logische Schlüsse auf mögliche Einflussfaktoren gebildet werden.

1.2 Verwandte Arbeiten

Das Feld der Komplexitätsbestimmung von Software besteht bereits ähnlich lange wie die Softwareentwicklung selbst. Erste Komplexitätsmaßzahlen, wie z.B. die zyklomatische Komplexität von McCabe[7] wurden in den sechziger und siebziger Jahren des zqwanzigsten Jahrhunderts entwickelt [8]. Also ist anzunehmen, dass auch zu der Verifizierung dieser Arbeiten bereits eine Vielzahl an theoretischen Abhandlungen existieren. Durch eine Literaturrecherche konnten einige Arbeiten zu der Verifizierung von Softwarekomplexitätsmetriken identifiziert werden. Diese werden im Folgenden zusammenfassend beschrieben.

In Kemerer 1987 werden verschiedene Komplexitätsmaßen auf 15 Projekte einer Firma angewendet. Dabei kommen die Codezeilen basierten Maßen SLIM und COCOMO, sowie die nicht-Codezeilen-basierten Maßen ESTIMACS und Function Points zum Einsatz[9]. Die Autoren kamen zu dem Ergebnis, dass die Metriken nur aussagekräftig sind, wenn sie auf die Projekte individuell kalibriert werden. Mit der Kalibrierung konnten Genauigkeitsraten von 88% erzielt werden[10].

In Rumreich/Kecskemety 2019 werden die Komplexitätsmetriken Anzahl Codezeilen, die zyklomatische Komplexität, die Halstead Komplexitätsmaßzahl und der Maintainability Index auf Projekte von Studenten angewendet, um so zu erfahren, ob eine Änderung der Lehrmethode die Komplexität der Projekte beeinflusst[11]

In Alenezi/Almustafa 2015 wird der generelle Verlauf von Softwarekomplexitätsmetriken untersucht. Dabei kann das sechste Lehmansche Gesetz bewiesen warden, nach dem die Komplexität einer Software über Zeit steigt[12].

[7] Vgl. McCabe 1976
[8] Vgl. Zuse 1991, S. 25 und Rubey/Hartwick 1968
[9] Vgl. Kemerer 1987, S. 2
[10] Vgl. Kemerer 1987, S. 12
[11] Vgl. Rumreich/Kecskemety 2019, S. 1
[12] Vgl. Alenezi/Almustafa 2015, S. 262

1.3 Forschungsbeitrag

Der Forschungsbeitrag dieser Arbeit liegt in der Beschreibung der Korrelation von mathematisch berechneten Komplexitätsmaßen mit Aufwandsabschätzungen von Experten. Dieser Arbeitsbeitrag stellt in dreierlei Hinsicht einen Zusatznutzen dar:

Zum einen können Softwarekomplexitätsmetriken für den spezifischen Anwendungsbereich des DIL ausgewählt und validiert werden. Das ermöglicht es der Abteilung, diese Maßzahlen in Zukunft für die kontinuierliche Analyse bestehender Projekte sowie für die initiale Beurteilung neuer Projekte zu verwenden.

Zweitens werden die Metriken nicht nur im Kontext des DIL, sondern auch für die Allgemeinheit validiert. Zu den hier behandelten Projekten existieren noch keine öffentlichen Softwarekomplexitätsanalysen. Mit dieser Arbeit können die Metriken also besser beurteilt werden.

Drittens wurde im Laufe dieser Arbeit festgestellt, dass eine Abweichung der Metriken von den Aufwandsabschätzungen ein Indikator für Prozessereignisse der Softwareentwicklung seien könnte. Diese Arbeit könnte also die Basis für ein System zur kontinuierlichen Verbesserung agiler Softwareentwicklungspraktiken darstellen.

1.4 Methodisches Vorgehen

Als Abschlussarbeit eines Studiums wird sich diese Arbeit auch an den Methodiken der Wirtschaftsinformat (WI) orientieren. Die WI ist in ihrer Erkenntnisgewinnung methodenpluralistisch aufgestellt[13]. Ihr instrumentales Portfolio beinhaltet sowohl Methodiken aus den Real-, den Formal- sowie den Ingenieurswissenschaften[14].

Unter einer Methode wird generell eine spezielle Vorgehensweise verstanden. Sie zeichnet sich durch ein Regelsystem von Untersuchungsinstrumenten aus. Ist diese Methode als wissenschaftlich zu klassifizieren, müssen diese Regeln auch intersubjektiv nachvollziehbar sein[15].

In der WI lassen sich zwei erkenntnistheoretische Ansätze herausstellen: Das konstruktionswissenschaftliche Paradigma strebt nach dem Schaffen und Evaluieren von Informationssystemen in Form von Modellen, Methoden und Softwaresystemen[16]. Es weist z.B. Vorgehen der Informationssystemsgestaltung, wie das Prototyping auf[17]. Dem gegenüber steht das behavioristische und verhaltenswissenschaftliche Paradigma. Nach diesem Paradigma wird in der WI das Verhalten und die Auswirkungen von bestehenden Informationssystemen untersucht[18].

[13] Vgl. Wilde/Hess 2006, S. 1
[14] Vgl. Wilde/Hess 2006, S. 1
[15] Vgl. Wilde/Hess 2006, S. 1f
[16] Vgl. Wilde/Hess 2006, S. 2
[17] Vgl. Wilde/Hess 2006, S. 3 und Simon 2019
[18] Vgl. Wilde/Hess 2006, S. 3

In dieser Arbeit soll nach dem behavioristischen Paradigma eine Erkenntnis gewonnen werden[19]. Der bestehende Sachzusammenhang der Korrelation von mathematischen Komplexitätsmetriken mit subjektiven Aufwandsabschätzungen in agilen Projekten soll untersucht werden. Aus dieser Untersuchung soll induktiv auf einen Gesamtzusammenhang bzw. eine Gesetzesmäßigkeit geschlossen werden.

Als Hypothese zu dieser Gesetzesmäßigkeit wird in dieser Arbeit eine eingeschränkte Korrelation zwischen den Aufwandsabschätzungen und den Codekomplexitätsmetriken vermutet. Es ist grundsätzlich anzunehmen, dass eine Erhöhung der mathematischen Komplexität auch in einer Erhöhung des geschätzten Aufwandes widergespiegelt wird. Jedoch sind auch Störfaktoren dieser Korrelation abzusehen.

Die Validierung einer Metrik durch die Untersuchung ihrer Korrelation mit einer anderen Größe ist nach Zuse, Fenton und Bowl ein verbreiteter Ansatz[20]. Mit der Untersuchung dieser Korrelation können die Komplexitätsmetriken also im Kontext des DIL validiert werden. Eine genaue Korrelation kann und soll in dieser Arbeit aufgrund der zu erwartenden Störfaktoren und des geringen Umfangs der Untersuchung aber nicht berechnet werden[21]. Vielmehr soll eine unvollständige Theorie in Form einer Näherungsangabe als ceteris-paribus Hypothese aufgestellt werden[22].

Trotz ihres vergleichsweisen jungen Alters bietet die WI eine üppige Bandbreite an Methoden der Erkenntnisgewinnung[23]. Gerade die Methodik der Fallstudienforschung erlebt einen stetigen Anstieg in Popularität[24]. Insbesondere für eine beschreibende Forschung, wie sie in dieser Arbeit angestrebt wird, sei die Fallstudienmethodik geeignet[25]. In einer Fallstudie wird ein Phänomen in seinem natürlichen Kontext beschrieben. Es wird eine geringe Anzahl von Fällen intensiv sowohl mit qualitativen als auch quantitativen Analysemethoden untersucht[26]. Es werden verschiedene Datentypen gesammelt und diese in Verbindung zueinander und zu der Hypothese gebracht. Zum Herstellen dieser logischen Verbindungen steht eine Reihe von Werkzeugen zur Verfügung. Insbesondere die Zeitreihenanalyse findet in dieser Arbeit Anwendung[27]. Der Aufbau der Fallstudie[28] wird in Kapitel 4 weiter beschrieben.

[19] Vgl. Wilde/Hess 2006, S. 3
[20] Vgl. Zuse 1991, S. 561f
[21] Vgl. Jones 2008, S. 449
[22] Vgl. Wilde/Hess 2006, S. 3
[23] Vgl. Heinrich 2005, S. 113
[24] Vgl. Yin 2014, S. 22
[25] Vgl. Dubé/Paré 2003, S. 607
[26] Vgl. Göthlich 2003, S. 7
[27] Vgl. Göthlich 2003, S. 6
[28] Vgl. Göthlich 2003, S. 8ff

1.5 Aufbau der Arbeit

Der Aufbau dieser Arbeit orientiert sich an dem Aufbau ähnlicher Arbeiten[29] und soll so eine zielgerichtete und übersichtliche Erfassung der Maßzahlen ermöglichen.

Zunächst wird das Thema der Softwarekomplexität aus einem generellen Blickwinkel betrachtet (Kapitel 2). Dabei wird der Begriff Softwarekomplexität zunächst definiert und dann erläutert (Kapitel 2.1). In einem zweiten Schritt werden verschiedene Methoden zur Messung von Softwarekomplexität begründet ausgewählt und erklärt (Kapitel 2.3).

Als Gegenstück zu den Softwarekomplexitätsmetriken werden in Kapitel 3 die Komplexitätsabschätzungen der Experten erläutert. Dabei wird insbesondere auf den Kontext der Komplexitätsabschätzungen in der agilen Entwicklung der Projekte Bezug genommen.

Nach der Definition der beiden Untersuchungsgrößen wird in Kapitel 4 der Aufbau der Fallstudienforschung beleuchtet. Es wird ein formales Forschungsprotokoll definiert. Insbesondere wird ein Analysealgorithmus zur automatisierten Untersuchung der Projekte vorgestellt.

Ab Kapitel 5 findet die praktische Untersuchung der Komplexitätsmetriken statt. Es werden die einzelnen Projekte als Fälle vorgestellt. Jeweils wird die Datenerhebung beschrieben und mit der zuvor erläuterten Forschungsmethodik ausgewertet. In jedem Projekt wird ein Fazit zu der Korrelation der Metriken gezogen. In einem letzten Schritt werden gesammelte Zusatzdaten aus den Projekten hinzugezogen, um das Forschungsergebnis zu erklären und auch mögliche Störfaktoren aufgezeigt.

Eine gesammelte Analyse aller Ergebnisse findet in Kapitel 6 statt. Hier wird versucht, aus den Ergebnissen der einzelnen Projekte einen Schluss auf eine generelle Korrelation zu ziehen.

Zuletzt wird in Kapitel 7 ein Fazit der Arbeit gegeben. Dabei wird insbesondere auf mögliche Kritik an den Forschungsergebnissen eingegangen (Kapitel 7.1) und ein Ausblick auf weitere Entwicklungen gegeben (Kapitel 7.2).

[29] Vgl. Alenezi/Almustafa 2015, S. 260

2 Softwarekomplexitätsmaße

Die Analyse der Softwarekomplexität ist ein Teil der statischen Sourcecode-Analyse. In der Code-Analyse wird im Nachhinein (a posterio[30]) anhand von statischen Analysen einer Stichprobe zu einem bestimmten Zeitpunkt[31] des Sourcecodes festgestellt, ob die Software vorher definierten Qualitätsanforderungen entspricht[32]. Diese Analyse kann sowohl automatisiert als auch manuell erfolgen und wird im Gegensatz zu Verfahren der konstruktiven Qualitätssicherung auf bereits vorhandene Software angewendet[33].

Als Teil der statischen Code-Analyse ist die Messung der Softwarekomplexität ein Teil der Qualitätssicherung der Software[34]. Dabei sollen oft unsichtbare Eigenschaften der Software sichtbar und quantitativ messbar gemacht werden[35][36]. Der Qualität von Software können verschiedene Teilbereiche untergeordnet werden. Je nach Definition gehören hierzu unter anderem Funktionalität, Laufzeit, Zuverlässigkeit, Benutzbarkeit, Wartbarkeit, Transparenz, Übertragbarkeit und Testbarkeit[37][38]. Gerade auf die Qualitätsmerkmale Wartbarkeit und Testbarkeit hat die Komplexität der Software einen direkten Einfluss. So steigt der Aufwand des Wartens und Testens mit dem Umfang und der Komplexität der Software.

Im Lebenszyklus einer Software kann die Komplexitätsuntersuchung als Teil der Qualitätssicherung sowohl an das Ende der Entwicklungsphase gestellt werden als auch kontinuierlich parallel zu der Weiterentwicklung stattfinden.

2.1 Definitionen

Eine Betrachtung von Softwarekomplexitätsmaßen setzt zunächst eine genaue Definition dieser voraus. Softwarekomplexitätsmetriken sind Metriken zur Messung der Komplexität einer Software. Diese drei Teile der Definition von Softwarekomplexitätsmetriken werden im Folgenden definiert.

Laut dem Institute of Electrical and Electronics Engineers (IEEE) Standard Glossar der Softwareentwicklung besteht *Software* aus den Computerprogrammen, Prozeduren und gegebenenfalls der Dokumentation und den Daten, die den Betrieb eines Computersystems betreffen[39]

[30] Vgl. Hoffmann 2013, S. 261
[31] Vgl. Ebert 1996, S. 86
[32] Vgl. Ebert 1996, S. 261
[33] Vgl. Hoffmann 2013, S. 261
[34] Vgl. Hoffmann 2013, S. 261
[35] Vgl. Hoffmann 2013, S. 261
[36] Vgl. Zuse 1991, S. 561
[37] Vgl. Hoffmann 2013, S. 22f
[38] Vgl. Liggesmeyer 2009, S. 245
[39] Vgl. *IEEE Standard Glossary of Software Engineering Terminology* 2022, S. 66

Die Definition von *Komplexität* wird allgemein als schwierig erachtet[40]. Aber auch hier schlägt das IEEE Standard Glossar eine allgemein anerkannte Definition vor: „[Complexity is] the degree to which a system or component has a design or implementation that is difficult to understand and verify“[41]. Nach dieser Definition ist Komplexität ein Maß dafür, wie schwierig eine Software zu verstehen und zu validieren ist. Fenton und Jones bauen auf dieser Definition auf und schlagen vier Arten von Komplexität vor: 1. Die Komplexität des Problems, welches die Software zu lösen versucht, 2. Die Komplexität der Algorithmen der Software, 3. Die Komplexität der Struktur der Software und 4. Die kognitive Komplexität der Software [42]. Nach dieser Kategorisierung bestimmen die, in dieser Arbeit behandelten Metriken die Komplexität der Struktur der Software (3). Weiter lässt sich zwischen inter- und intra-modularen Komplexitätsmaßen unterscheiden. Hier werden ausschließlich intra-modulare Komplexitätsmaßen behandelt. Diese Messen die Komplexität einzelner Programmteile[43].

Im Kontext der Softwaremetrie werden Metriken als ein Messystem bzw. ein Verfahren zum Quantifizieren von Eigenschaften von Software definiert [44]. Im Deutschen werden sie oft als Synonym für Maß bzw. Maßzahl genutzt. Im Englischen findet zwischen den Begriffen „metric“ und „measure“ eine genauere Unterscheidung statt: Eine Metrik sei eine Funktion, die als Eingabe Daten eines Gegenstandes verwendet und hieraus eine Zahl zur Quantifizierung dieser Eigenschaft(en) liefert[45]. Ein Maß (measure) sei dahingegen die Konkrete Anwendung dieser Metrik[46]. Aufgrund dieser Unterscheidung erscheint der Begriff Metrik für die, in dieser Arbeit behandelten Verfahren als passender.

Zusammengefasst werden Softwarekomplexitätsmetriken in dieser Arbeit als Verfahren zur Quantifizierung der Vielschichtigkeit der Struktur eines Computerprogramms definiert.

2.2 Kritik

Die Praktik der Softwarekomplexitätsmetrie wird allgemein stark kritisiert.

Zum einen besteht Kritik an dem quantitativen Erfassen subjektiver Softwareeigenschaften im Generellen. So können Metriken nicht *direkt* messen, was für Menschen als subjektive Komplexität wahrgenommen wird, auch wenn das Verwenden von mehreren Maßzahlen hilft, eine robustere Perspektive zu schaffen[47].

[40] Vgl. Jones 2008, S. 335 und 627
[41] Vgl. *IEEE Standard Glossary of Software Engineering Terminology* 2022, S. 18
[42] Vgl. Fenton/Pfleeger 2003, S. 258 und Jones 2008, S. 449
[43] Vgl. Zuse 1991, S. 7ff
[44] Vgl. Dumke 1994, S. 35ff, Ebert 1996, S. 4ff, Augsten 2022 und *IEEE Standard for a Software Quality Metrics Methodology* 2022, S. 2f
[45] Vgl. *IEEE Standard for a Software Quality Metrics Methodology* 2022, S. 3
[46] Vgl. *IEEE Standard for a Software Quality Metrics Methodology* 2022, S. 2
[47] Vgl. Rumreich/Kecskemety 2019, S. 2

Weiter werden auch die Komplexitätsmetriken im Speziellen angezweifelt: Viele Metriken vereinen mehrere, oft konfliktäre Messziele[48]. Dadurch wird ihre Aussagekraft verwässert. Nicht nur an dem mathematischen Aufbau der Metriken, sondern auch an den Implementierungen dieser gibt es Kritik. So liefern verschiedene Implementierungen derselben Metrik oft verschiedene Ergebnisse[49]. Zusätzlich wurden die klassischen Metriken, wie z.B. die zyklomatische Komplexität für imperative Programmiertechniken entwickelt. Das lässt sich darauf zurückführen, dass zu ihrem Entwicklungszeitpunkt die imperative Programmierung noch am weitesten verbreitet war[50]. Heutzutage sind andere Programmiertechniken, wie z.B. die objekt-orientierte Programmierung verbreiteter. Diese neuen Programmiertechniken wurden jedoch in den klassischen Metriken nicht berücksichtigt[51]. Ein Beispiel dieser Problematik sind die Vererbungsmechanismen in objekt-orientierten Programmiersprachen. Eine Vererbung erhöht z.B. in der klassischen Metrik der zyklomatischen Komplexität nicht die Komplexität, was an dieser Stelle die Korrelation von Code-Größe und der Komplexität des Codes aushebelt[52].

Ferner ist auch der Zusammenhang zwischen Softwaremetriken und externen Softwareeigenschaften, wie z.B. der Fehleranfälligkeit umstritten[53]. Hier konnten sowohl Studien gefunden werden, die für eine Korrelation sprechen, also auch Studien, die dagegensprechen. In Ebert 1996 konnte z.B. ein Zusammenhang zwischen der gemessenen Komplexität einer Aufgabe und den dabei entstandenen Fehlern nachgewiesen werden[54]. Im Gegensatz dazu konnten in Revilla 2007 keine Zusammenhänge zwischen internen Softwaremetriken und externen Eigenschaften von Software festgestellt werden[55].

Als mögliches Fazit aus diesen Studien wird in dieser Arbeit vorschlagen, dass die Anwendbarkeit der Maßzahlen vom jeweiligen Kontext des Softwareprojektes und von dem erwarteten Ergebnis abhängt. Dieses Fazit unterstreicht noch einmal die Relevanz dieser Arbeit. So kann mit dem Ergebnis dieser Arbeit evaluiert werden, ob und welche Softwaremaßzahlen im Kontext des DIL Ratingen anwendbar sind.

2.3 Eine Auswahl von Softwarekomplexitätsmaßzahlen

Trotz der umfangreichen Kritik an der Vermessung von Software wurden in den letzten Jahrzehnten eine Vielzahl von Maßzahlen entwickelt.

In dieser Arbeit werden insgesamt vier Metriken für die Komplexität von Software betrachtet. Zunächst bietet die Anzahl an logischen Codezeilen einen ersten Eindruck der Komplexität. Sie ist aber noch eher ein Umfangsmaß als ein Komplexitätsmaß. Weiter werden die zyklomatische

[48] Vgl. Fenton/Pfleeger 2003, S. 322
[49] Vgl. Rumreich/Kecskemety 2019, S. 2
[50] Vgl. Hoffmann 2013, S. 277
[51] Vgl. Hoffmann 2013, S. 277
[52] Vgl. Hoffmann 2013, S. 277
[53] Vgl. Jones 2008, S. 627
[54] Vgl. Ebert 1996, S. 65
[55] Vgl. Revilla 2007, S. 203 und 208

Komplexität von Thomas McCabe, sowie die Softwaremaßzahlen von Halstead behandelt. Als letzte Maßzahl wird die Einrückungskomplexität der Autoren Hindle, Godfrey, und Holt betrachtet.

Die Auswahl der ersten drei Komplexitätsmaßen beruft sich auf ihre generelle Popularität in der Softwaremetrie. Im Rahmen einer umfangreichen Literaturrecherche konnten für diese Metriken die meisten Referenzen gefunden werden. Die Maßzahl der logischen Codezeilen wird von Zuse91 als ein wichtiges Maß zum Bestimmen des Umfangs und der Komplexität einer Softwareanwendung eingestuft[56]. Diese Position wird auch von Sato u. a. 2006 und Alenezi/Almustafa 2015 bekräftigt. Die zyklomatische Komplexität von McCabe wird ebenfalls von Zuse als eine der bekanntesten Maßzahlen eingestuft[57]. Weiter bekräftigt fentonSoftwareMetricsRigorous2003, dass diese Maßzahl zum Messen der Softwarekomplexität geeignet sei[58]. Auch die vier Autoren Revilla[59], Jones[60], Sato[61] und Alenezi[62] verweisen auf die zyklomatische Komplexität als Maßzahl für die Komplexität einer Applikation. Jones sagt dabei zusätzlich aus, dass diese Komplexitätsmaßzahl ein besonders breites Anwendungsspektrum bedienen könne[63]. Auch die Softwaremaßzahlen von Halstead werden in der Literatur häufig referenziert. Zuse und Revilla bekräftigen die Verbreitung dieser Maßzahlen[64]. Zusätzlich sagt Fenton aus, dass die Maßzahlen geeignet zum Messen der Softwarekomplexität seien[65].

Zusätzlich wird die Einrückungskomplexität betrachtet, da sie einen, im Gegensatz zu den anderen Maßzahlen methodisch sehr abweichenden Ansatz verfolgt. So betrachtet sie im Gegensatz zu den anderen Maßzahlen nicht die Aufteilung auf Codezeilen oder den Inhalt des Codes, sondern die Einrückung der einzelnen Codezeilen.

Neben diesen vier Maßzahlen bestehen auch noch eine Vielzahl an weiteren Messungsmethoden, die in dieser Arbeit aber keine weitere Betrachtung finden sollen. Zum Beispiel wurden die gewichtete Anzahl an Methoden pro Klasse (WMC)[66], die Anzahl an Kommentaren[67], die Größe der Klasse[68] und die Anzahl an Testcodezeilen[69] nicht weiter betrachtet.

Zwischen drei der ausgewählten Maßzahlen wurden bereits Korrelationen nachgewiesen. Zunächst besteht eine starke Korrelation zwischen der Anzahl an logischen Codezeilen und dem Aufwand nach Halstead[70]. Eine noch stärkere, linear-stabile Korrelation lässt sich zwischen der Anzahl logischer Code Zeilen und der zyklomatischen Komplexitätsmaßzahl nachweisen. Diese Korrelation

[56] Vgl. Zuse 1991, S. 145
[57] Vgl. Zuse 1991, S. 145
[58] Vgl. Fenton/Pfleeger 2003, S. 31
[59] Vgl. Revilla 2007, S. 203
[60] Vgl. Jones 2008, S. 335, 627 und 449
[61] Vgl. Sato u. a. 2006
[62] Vgl. Alenezi/Almustafa 2015
[63] Vgl. Jones 2008, S. 335, 627 und 449
[64] Vgl. Zuse 1991, S. 145 und Revilla 2007, S. 203
[65] Vgl. Fenton/Pfleeger 2003, S. 31
[66] Vgl. Sato u. a. 2006
[67] Vgl. Revilla 2007, S. 207
[68] Vgl. Sato u. a. 2006
[69] Vgl. Sato u. a. 2006
[70] Vgl. Jones 2008, S. 627

konnte über Programmierer*innen, Programme, Sprachen und Programmierparadigma hinweg bewiesen werden[71].

Die vier Komplexitätsmetriken werden nun der Reihe nach genauer erläutert.

2.3.1 Logische Codezeilen

Als erste hier vorgestellte Metrik weist die Anzahl der logischen Codezeilen (SLOC) den geringsten Berechnungsaufwand auf. Sie wird teilweise auch als Anzahl der Codezeilen (LOC) bezeichnet[72]. Dabei werden die Zeilen an Sourcecode in der Software berechnet[73]. Die Metrik lässt sich somit ohne die Unterstützung komplexer Algorithmen berechnen und lässt sich auf nahezu alle Programmiersprachen anwenden[74]. Eine Ausnahme bilden hier grafische Programmiersprachen.

In ihrer Aussagekraft wird die Anzahl von Codezeilen allgemein als begrenzt eingestuft[75]. Unter anderem der Programmierstil und die Programmiersprache haben einen Einfluss auf die Anzahl der Codezeilen (Hoffmann 2013:264, Hoffmann 2013:264). Gleichzeitig erfüllt sie jedoch die Assoziativität, Kommutativität und Monotonität[76] als wesentliche Qualitätskriterien für Maßzahlen.

Eine Weiterentwicklung der LOC-Metrik ist u.a. die NCSS-Metrik (Non Commented Source Statements). Sie misst genauso wie die LOC-Metrik die Anzahl an Codezeilen, ignoriert dabei aber alle Kommentarzeilen[77]. Zusätzlich bestehen auch empirisch ermittelte Sprachfaktoren, die es ermöglichen, die Ergebnisse von LOC und NCSS Metriken vor ihrer Weiterverarbeitung zu gewichten[78].

2.3.2 Zyklomatische Komplexität

Die zyklomatische Komplexität wurde von Tom McCabe entwickelt und zuerst in seinem Aufsatz „A Complexity Metric“[79] veröffentlicht[80]. Die zyklomatische Komplexität gehört zu den Kontrollflussmetriken[81]. In der Praxis wird sie von vielen Firmen entwicklungsbegleitend kontinuierlich erhoben, um vor problematischen Programmkomponenten frühzeitig zu warnen. Dabei fließt sie oft auch in die Abnahmekriterien für Software ein[82].

[71] Vgl. Jones 2008, S. 627 und Jay u. a. 2009, S. 137
[72] Vgl. Hoffmann 2013, S. 263
[73] Vgl. Rumreich/Kecskemety 2019, S. 2
[74] Vgl. Hoffmann 2013, S. 263
[75] Vgl. Hoffmann 2013, S. 264 und Rumreich/Kecskemety 2019, S. 2
[76] Vgl. Zuse 1991, S. 142
[77] Vgl. Hoffmann 2013, S. 264
[78] Vgl. Hoffmann 2013, S. 264
[79] Vgl. McCabe 1976
[80] Vgl. Sneed/Seidl/Baumgartner 2010, S. 185
[81] Vgl. Ebert 1996, S. 88
[82] Vgl. Hoffmann 2013, S. 275

Die zyklomatische Komplexität ist definiert als eine Maßzahl für die Kontrollflusskomplexität eines Programmes[83]. Sie misst die Anzahl von linear unabhängigen Pfaden auf dem Kontrollflussgraphen eines Programmes und liefert so ein normalisiertes Komplexitätsmaß[84].

Zur Berechnung der zyklomatischen Komplexität wird der Programmcode zunächst in einen Graphen aus Knoten und Kanten umgewandelt[85]. Dazu wird der Quelltext in einen Syntaxbaum umgewandelt. Diese Baumstruktur spiegelt die Struktur des Codes wider. Aus dem Syntaxbaum wird ein Kontrollflussgraph der Anwendung konstruiert. Der Kontrollflussgraph besteht aus den Anweisungen und Verzweigungen des Programms und stellt alle möglichen Ablaufwege durch das Programm dar[86]. Der Umwandlungsprozess wird in Abbildung .. beschrieben.

GRAFIK

Aus den Eigenschaften dieses Kontrollflussgraphen kann nun die zyklomatische Komplexität mit Mitteln der Graphentheorie[87] berechnet werden. Als relevante Größen kommen dabei die Anzahl der Kanten des Graphen (E), die Anzahl der Knoten (N), sowie die Anzahl unabhängiger Teilgraphen (p) zum Einsatz. Die Anzahl unabhängiger Teilgraphen entspricht der Anzahl unabhängiger Module bzw. Funktionen in dem Programm. Wird ein Programm mit mehreren Modulen untersucht, werden die Kanten und Knoten der Kontrollflussgraphen der Module aufsummiert[88].

Die Formel für die zyklomatische Komplexität lautet wie folgt:

V(Z) = |E|-|N| + 2p

Die zyklomatische Komplexität (V(Z)) entspricht also der Anzahl an Kanten (E) minus der Anzahl an Knoten (N) plus der doppelten Anzahl der zusammenhängenden Einzelgraphen (p) (Hoffmann 2013:273, Jones 2008:335, Hoffmann 2013:273, Ebert 1996:88, Sneed et al 2010:185). Also ist sie stets um eins größer als die Anzahl an Verzweigungen in einem Programm (Hoffmann 2013:274).

In der konkreten Anwendung wird die zyklomatische Komplexität oft als Indikator für die Wartbarkeit des Code und für die Produktivität in der Weiterentwicklung des Codes verwendet[89]. Zusätzlich liefert sie die obere Grenze der Anzahl an Tests, die für eine vollständige Testabdeckung des Quelltextes benötigt werden.

Die zyklomatische Komplexität wurde in der Fachliteratur vielfach zitiert und vor allem auch vielfach kritisiert.

[83] Vgl. Rumreich/Kecskemety 2019, S. 2 und Jones 2008, S. 335
[84] Vgl. Rumreich/Kecskemety 2019, S. 2
[85] Vgl. Sneed/Seidl/Baumgartner 2010, S. 185 und Jones 2008, S. 335
[86] Vgl. Räcke 2017
[87] Vgl. Hoffmann 2013, S. 273
[88] Vgl. Hoffmann 2013, S. 275f
[89] Vgl. Jones 2008, S. 336

Zunächst wird allgemein kritisiert, dass die Maßzahl nur die Komplexität der Ablauflogik des Codes und nicht die Komplexität des gesamten Codes misst[90]. Interne Eigenschaften der Knoten[91], die Komplexität im Datenfluss[92] sowie die Verschachtelung von Modulen werden ignoriert[93]. Also lässt sich sagen, dass die Maßzahl für prozeduralen Code eine gewisse Bedeutung hat, in objektorientiertem Code jedoch nur die Komplexität der einzelnen Methodenkörper misst[94].

Insgesamt wird geschlussfolgert, dass die zyklomatische Komplexität nur mit Vorsicht anzuwenden sei und in Relation zu anderen Maßzahlen gesetzt werden müsse[95]. Genau aus diesem Grund wird sie in dieser Arbeit in den Kontext der agilen Aufwandsabschätzungen und in Relation zu anderen Maßzahlen gesetzt.

2.3.3 Halstead Metriken

Als dritte Komplexitätsmetrik werden die Halstead Metriken herangezogen. Bei diesen Metriken wendet der Autor Maurice Howard Halstead[96] Elemente der Kommunikationstheorie auf die Programmierung an[97].

Die Berechnung der Halstead Metriken wird im folgenden an einem Beispiel erklärt[98]. Das Beispiel behandelt folgenden Code:

```
1 main()
2 {
3 int countOne, countTwo, countThree, average;
4 scanf(``%d %d %d'', &countOne, &countTwo, &countThree);
5 average = (countOne + countTwo + countThree) / 3;
6 printf(``average = \%d'', average);
7 }
```

Für die Berechnung seiner Metriken zählt Halstead zunächst die Grundelemente der Programmiersprache, nämlich die Operatoren und Operanden[99]. Operatoren führen Operationen auf Operanden durch. Beispiele für Operatoren sind arithmetische Operatoren (z.B. Addition) Vergleichsoperatoren und Zuweisungsoperatoren. Operanden sind in der Regel Datenelemente wie Zahlen oder Text.

In diesem Beispiel sind die unterschiedlichen Operatoren: main, (), {}, int, scanf, &, =, +, /, printf, „ ;

[90] Vgl. Sneed/Seidl/Baumgartner 2010, S. 186
[91] Vgl. Hoffmann 2013, S. 73 und Rumreich/Kecskemety 2019, S. 2
[92] Vgl. Rumreich/Kecskemety 2019, S. 2
[93] Vgl. Zuse 1991, S. 89ff
[94] Vgl. Sneed/Seidl/Baumgartner 2010, S. 186
[95] Vgl. Fenton/Pfleeger 2003, S. 52
[96] Vgl. Halstead 1979
[97] Vgl. Sneed/Seidl/Baumgartner 2010, S. 183
[98] Vgl. Sneed/Seidl/Baumgartner 2010, S. 184
[99] Vgl. Sneed/Seidl/Baumgartner 2010, S. 183 und Rumreich/Kecskemety 2019, S. 2

Die unterschiedlichen Operanden sind: countOne, countTwo, countThree, average, "%d %d %d", 3, ävg = %d"

Zu den Operatoren und Operanden werden folgende Zahlen erfasst:

- n1: Die Anzahl an unterschiedlichen Operatoren (12)
- n2: Die Anzahl an unterschiedlichen Operanden (7)
- N1: Die Anzahl insgesamt vorkommender Operatoren (27)
- N2: Die Anzahl insgesamt vorkommender Operanden (15)

Aus diesen Zahlen können nun die Halstead Metriken berechnet werden:

Der Wortschatz eines Programms ergibt sich aus der Summe der unterschiedlichen Operatoren und Operanden:

Wortschatz (n) = Operatoren (n1) + Operanden (n2) 19 = 12 + 7

Die Länge des Programmes ergibt sich aus der Summe der insgesamt vorkommenden Operanden und Operatoren :

Länge (N) = Operatorenvorkommnisse (N1) + Operandenvorkommnisse (N2) 42 = 27 + 15

Weiter Berechnet Halstead die Größe des Programmcodes durch einen Logarithmus:

*Größe (Volume / V) = Länge (N) * log2 (Wortschatz (n)) 53,71 = 42 * log(19)*

Für die Sprachkomplexität setzt er jeweils die Operatoren ins Verhältnis zu den Operatorenvorkommnissen und die Operanden ins Verhältnis zu den Operandenvorkommnissen. Beide Werte werden miteinander multipliziert:

*Komplexität = (Operatoren / Operatorenvorkommnisse) * (Operanden / Operandenvorkommnisse) 0,207 =ca (12 / 27) * (7 / 15)*

Zuletzt berechnet er den Aufwand als Quotient aus Programmgröße und Komplexität. Je komplexer oder größer ein Programm ist, desto länger dauere, es dieses Programm zu schreiben:

Aufwand (E) = Größe / Komplexität 259,47 = 53,71 / 0,207

Die von Halstead vorgeschlagenen Metriken wurden vielfach kritisiert. Er hat die Berechnungen nie empirisch untermauert und in späteren empirischen Studien wurden sie sogar widerlegt[100]. Insbesondere sei zu kritisieren, dass die Codegröße, Komplexität und der Aufwand nach Halstead nicht das Kriterium der Monotonität erfüllen. Sie können also nicht auf einer Verhältnisskala verwendetet werden und nur die Berechnung von Medianwerten sei sinnvoll[101]. Eine zusätzliche Einschränkung sei, dass die Halstead Metriken die Berechnungskomplexität in den Vordergrund

[100] Vgl. Sneed/Seidl/Baumgartner 2010, S. 185
[101] Vgl. Zuse 1991, S. 142

rücken und keine Aussage über den Kontrollfluss der Anwendung treffen[102]. Insgesamt seien sie also kritisch anzusehen[103].

2.3.4 Einrückungskomplexität

Als letztes Komplexitätsmaß wird hier die Einrückungskomplexität vorgestellt. Sie wurde von Abram Hindle, Michael W. Godfrey und Richard C. Holt an der University of Waterloo entwickelt und zuerst mit ihrer Arbeit „Reading Beside the Lines: Indentation as a Proxy for Complexity Metrics “ auf der sechzehnten internationalen IEEE Konferenz zum Verständnis von Computerprogrammen vorgestellt[104]. Die Komplexitätsmaßzahl ist im Vergleich zu den anderen hier behandelten Metriken verhältnismäßig neu, wurde aber bereits in Arbeiten von Lalouche et. al[105][106] und Landman et. al[107] referenziert.

Die Einrückungskomplexität soll nun erklärt werden. In den meisten Programmiersprachen werden bestimmte Zeilen zur besseren Lesbarkeit des Codes eingerückt. Die Einrückungskomplexität macht sich genau diesen Umstand zu Nutzen und verwendet die Einrückungen der Codezeilen als Indikator für Komplexität[108].

In prozeduralen Code, wie z.B. in C, kann die Einrückung von Codezeilen Kontrollstrukturen, wie Verzweigungen und Schleifen anzeigen. In objektorientierten Sprachen, wie C++, Java und JavaScript kann die Einrückung eine Verkapselung in Form von Klassen, Subklassen oder Methoden anzeigen. In funktionalen Sprachen wie OCaml und Lisp zeigt die Einrückung einen neuen Handlungskontext, neue Funktionen oder einen neuen Ausdruck. In jedem dieser drei Typen von Programmiersprachen sind in der Regel Stellen im Code eingerückt, die die Komplexität des Codes erhöhen. Also lasse sich schlussfolgern, dass die Menge der Einrückungen ein Maß der Komplexität eines Programmes sei.

Zur Verifizierung dieser These setzten die Autoren die Einrückungskomplexität in ein Verhältnis zu den älteren Metriken wie der zyklomatischen Komplexität und den Halstead Metriken. Bei der zyklomatischen Komplexität erhöhen z.B. Verzweigungen die Komplexität. Diese werden in den meisten Programmiersprachen durch eine Einrückung der Codezeilen gekennzeichnet. Also stünde die Vermutung einer Korrelation der Metriken nahe[109]. Diese Korrelation konnte von den Autoren in einer Studie von 278 populären Open Source Projekten erfolgreich nachgewiesen werden. Damit sei die Validität der Einrückungskomplexität als Komplexitätsmaß von Software nachgewiesen.

[102] Vgl. Rumreich/Kecskemety 2019, S. 2
[103] Vgl. Sneed/Seidl/Baumgartner 2010, S. 185
[104] Vgl. Hindle, A./Godfrey, M./Holt, R. 2008, S. 1
[105] Vgl. Gil, J. (/Lalouche 2016, S. 10
[106] Vgl. Gil, Y./Lalouche 2017, S. 7
[107] Vgl. Landman u. a. 2016, S. 6
[108] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 1
[109] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 2

Für die Berechnung der Einrückungskomplexität werden die Einrückungen der einzelnen Codezeilen im Quelltext betrachtet. Die Anzahl an physischen Einrückungen in einer Zeile bezeichnet zunächst die Anzahl an Leerzeichen bzw. Tabs, welche sich am Anfang einer Zeile befinden. Diese werden für jede Zeile berechnet. Anhand dieser Angaben wird auf der Ebene einer Datei berechnet, wie viele Leerzeichen bzw. Tabs einer logischen Einrückungsebene entsprechen[110]. In den meisten Projekten entsprechen vier Leerzeichen bzw. ein Tab einem Einrückungslevel[111]. Dieser Zusammenhang wird von den Autoren als das Einrückungsmodell einer Datei bezeichnet. Mit dem Einrückungsmodell kann nun die Anzahl an logischen Einrückungen für jede Datei berechnet werden. Die Anzahl dieser Einrückungen gibt für jede Zeile die Einrückungsebene an. Die Summe der logischen Einrückungen wird als die Einrückungskomplexität bezeichnet.

Ein wesentlicher Vorteil der Einrückungskomplexität ist ihre Sprachenunabhängigkeit. Im Gegensatz zu der zyklomatischen Komplexität und den Halstead Metriken setzt sie kein Verständnis der Grammatik einer Programmiersprache voraus[112][113].

Als zusätzlichen Vorteil führen die Autoren auf, dass die Einrückungskomplexität weniger Berechnungsschritte benötige als andere Komplexitätsmaßen. Demnach sei sie günstiger in der Durchführung als manche andere Metriken[114]. Vergleichsoperatoren und Zuweisungsoperatoren. Operanden sind in der Regel Datenelemente wie Zahlen oder Text.

In diesem Beispiel sind die unterschiedlichen Operatoren: main, (), {}, int, scanf, &, =, +, /, printf, „ ;

Die unterschiedlichen Operanden sind: countOne, countTwo, countThree, average, "%d %d %d", 3, ävg = %d"

Zu den Operatoren und Operanden werden folgende Zahlen erfasst:

- n1: Die Anzahl an unterschiedlichen Operatoren (12)
- n2: Die Anzahl an unterschiedlichen Operanden (7)
- N1: Die Anzahl insgesamt vorkommender Operatoren (27)
- N2: Die Anzahl insgesamt vorkommender Operanden (15)

Aus diesen Zahlen können nun die Halstead Metriken berechnet werden:

Der Wortschatz eines Programms ergibt sich aus der Summe der unterschiedlichen Operatoren und Operanden:

Wortschatz (n) = Operatoren (n1) + Operanden (n2) 19 = 12 + 7

[110] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 5
[111] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 9
[112] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 1
[113] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 2
[114] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 20f

Die Länge des Programmes ergibt sich aus der Summe der insgesamt vorkommenden Operanden und Operatoren :

Länge (N) = Operatorenvorkommnisse (N1) + Operandenvorkommnisse (N2) 42 = 27 + 15

Weiter Berechnet Halstead die Größe des Programmcodes durch einen Logarithmus:

*Größe (Volume / V) = Länge (N) * log2 (Wortschatz (n)) 53,71 = 42 * log(19)*

Für die Sprachkomplexität setzt er jeweils die Operatoren ins Verhältnis zu den Operatorenvorkommnissen und die Operanden ins Verhältnis zu den Operandenvorkommnissen. Beide Werte werden miteinander multipliziert:

*Komplexität = (Operatoren / Operatorenvorkommnisse) * (Operanden / Operandenvorkommnisse) 0,207 =ca (12 / 27) * (7 / 15)*

Zuletzt berechnet er den Aufwand als Quotient aus Programmgröße und Komplexität. Je komplexer oder größer ein Programm ist, desto länger dauere, es dieses Programm zu schreiben:

Aufwand (E) = Größe / Komplexität 259,47 = 53,71 / 0,207

Die von Halstead vorgeschlagenen Metriken wurden vielfach kritisiert. Er hat die Berechnungen nie empirisch untermauert und in späteren empirischen Studien wurden sie sogar widerlegt[115]. Insbesondere sei zu kritisieren, dass die Codegröße, Komplexität und der Aufwand nach Halstead nicht das Kriterium der Monotonität erfüllen. Sie können also nicht auf einer Verhältnisskala verwendetet werden und nur die Berechnung von Medianwerten sei sinnvoll[116]. Eine zusätzliche Einschränkung sei, dass die Halstead Metriken die Berechnungskomplexität in den Vordergrund rücken und keine Aussage über den Kontrollfluss der Anwendung treffen[117]. Insgesamt seien sie also kritisch anzusehen[118].

2.3.5 Einrückungskomplexität

Als letztes Komplexitätsmaß wird hier die Einrückungskomplexität vorgestellt. Sie wurde von Abram Hindle, Michael W. Godfrey und Richard C. Holt an der University of Waterloo entwickelt und zuerst mit ihrer Arbeit „Reading Beside the Lines: Indentation as a Proxy for Complexity Metrics “ auf der sechzehnten internationalen IEEE Konferenz zum Verständnis von Computerprogrammen vorgestellt[119]. Die Komplexitätsmaßzahl ist im Vergleich zu den anderen hier behandelten Metriken verhältnismäßig neu, wurde aber bereits in Arbeiten von Lalouche et. al[120][121] und Landman et. al[122] referenziert.

[115] Vgl. Sneed/Seidl/Baumgartner 2010, S. 185
[116] Vgl. Zuse 1991, S. 142
[117] Vgl. Rumreich/Kecskemety 2019, S. 2
[118] Vgl. Sneed/Seidl/Baumgartner 2010, S. 185
[119] Vgl. Hindle, A./Godfrey, M./Holt, R. 2008, S. 1
[120] Vgl. Gil, J. (/Lalouche 2016, S. 10
[121] Vgl. Gil, Y./Lalouche 2017, S. 7
[122] Vgl. Landman u. a. 2016, S. 6

Die Einrückungskomplexität soll nun erklärt werden. In den meisten Programmiersprachen werden bestimmte Zeilen zur besseren Lesbarkeit des Codes eingerückt. Die Einrückungskomplexität macht sich genau diesen Umstand zu Nutzen und verwendet die Einrückungen der Codezeilen als Indikator für Komplexität[123].

In prozeduralen Code, wie z.B. in C, kann die Einrückung von Codezeilen Kontrollstrukturen, wie Verzweigungen und Schleifen anzeigen. In objektorientierten Sprachen, wie C++, Java und JavaScript kann die Einrückung eine Verkapselung in Form von Klassen, Subklassen oder Methoden anzeigen. In funktionalen Sprachen wie OCaml und Lisp zeigt die Einrückung einen neuen Handlungskontext, neue Funktionen oder einen neuen Ausdruck. In jedem dieser drei Typen von Programmiersprachen sind in der Regel Stellen im Code eingerückt, die die Komplexität des Codes erhöhen. Also lasse sich schlussfolgern, dass die Menge der Einrückungen ein Maß der Komplexität eines Programmes sei.

Zur Verifizierung dieser These setzten die Autoren die Einrückungskomplexität in ein Verhältnis zu den älteren Metriken wie der zyklomatischen Komplexität und den Halstead Metriken. Bei der zyklomatischen Komplexität erhöhen z.B. Verzweigungen die Komplexität. Diese werden in den meisten Programmiersprachen durch eine Einrückung der Codezeilen gekennzeichnet. Also stünde die Vermutung einer Korrelation der Metriken nahe[124]. Diese Korrelation konnte von den Autoren in einer Studie von 278 populären Open Source Projekten erfolgreich nachgewiesen werden[125]. Damit sei die Validität der Einrückungskomplexität als Komplexitätsmaß von Software nachgewiesen.

Für die Berechnung der Einrückungskomplexität werden die Einrückungen der einzelnen Codezeilen im Quelltext betrachtet. Die Anzahl an physischen Einrückungen in einer Zeile bezeichnet zunächst die Anzahl an Leerzeichen bzw. Tabs, welche sich am Anfang einer Zeile befinden. Diese werden für jede Zeile berechnet. Anhand dieser Angaben wird auf der Ebene einer Datei berechnet, wie viele Leerzeichen bzw. Tabs einer logischen Einrückungsebene entsprechen[126]. In den meisten Projekten entsprechen vier Leerzeichen bzw. ein Tab einem Einrückungslevel[127]. Dieser Zusammenhang wird von den Autoren als das Einrückungsmodell einer Datei bezeichnet. Mit dem Einrückungsmodell kann nun die Anzahl an logischen Einrückungen für jede Datei berechnet werden. Die Anzahl dieser Einrückungen gibt für jede Zeile die Einrückungsebene an. Die Summe der logischen Einrückungen wird als die Einrückungskomplexität bezeichnet.

Ein wesentlicher Vorteil der Einrückungskomplexität ist ihre Sprachenunabhängigkeit. Im Gegensatz zu der zyklomatischen Komplexität und den Halstead Metriken setzt sie kein Verständnis der Grammatik einer Programmiersprache voraus [128].

[123] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 1
[124] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 2
[125] Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009
[126] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 5
[127] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 9
[128] Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 1 und Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 2

Als zusätzlichen Vorteil führen die Autoren auf, dass die Einrückungskomplexität weniger Berechnungsschritte benötige als andere Komplexitätsmaßen. Demnach sei sie günstiger in der Durchführung als manche andere Metriken[129].

[129]Vgl. Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009, S. 20f

3 Aufwandsabschätzungen agiler Projekte

In dieser Arbeit sollen Softwarekomplexitätsmaßen in Korrelation zu Aufwandsabschätzungen gesetzt werden. Für ein genaues Verständnis dieser Korrelation ist es auch von Nöten, die Aufwandsabschätzungen selbst genauer zu betrachten.

Die, in dieser Arbeit betrachteten Projekte werden alle in einer agilen Arbeitsweise realisiert. Im Rahmen dieser Vorgehensweise werden üblicherweise für jedes zu entwickelnde Feature Schätzungen des Aufwandes abgegeben. Für ein besseres Verständnis der Entstehung dieser Projekte wird im Folgenden zunächst die agile Arbeitsweise erläutert. Dann werden die Aufwandsabschätzungen der agilen Projekte genauer betrachtet und im Kontext dieser Arbeit erläutert.

3.1 Die agile Arbeitsweise in der Softwareentwicklung

Als die agile Arbeitsweise wird eine Reihe von iterativen Methoden[130] beschrieben, welche den Denk- und Arbeitsprozess von Softwareentwicklungsteams effizienter und effektiver gestalten sollen. Dabei werden alle Bereiche der traditionellen Softwareentwicklung, von dem Projektmanagement, über Softwaredesign und -architektur bis hin zur Prozessoptimierung betrachtet. Zu jedem dieser Bereiche bestehen Praktiken, die über Softwareprojekte hinweg möglichst einfach zu implementieren sein sollen[131].

Zusätzlich wird mit der agilen Arbeitsweise eine Mentalität beschrieben, bei der Planung, Design und Prozessoptimierung von dem gesamten Entwicklungsteam durchgeführt werden sollen[132].

3.2 Methoden der agilen Softwareentwicklung

Die agile Vorgehensweise wird in der Praxis in Form von verschiedenen Methoden umgesetzt. Diese Methoden definieren konkrete Abläufe und Verantwortlichkeiten. Zu dem Methoden gehören unter anderem Scrum, Kanban und das Scaled Agile Framework. Die in dieser Arbeit behandelten Projekte werden nach Scrum umgesetzt. Aus diesem Grund wird im folgenden Kapitel die Scrum Methodik erklärt.

[130] Vgl. Atlassian 2022
[131] Vgl. Stellman/Greene 2014, S. 2
[132] Vgl. Stellman/Greene 2014, S. 2

3.2.1 Ablauf

Zur Entwicklung eines Produktes nach Scrum befasst sich ein sog. Productowner zunächst mit den Wünschen der Stakeholder und hält diese in einem priorisierten Product-Backlog fest. Zu Beginn eines jeden Sprints wird in einem sog. Sprint Planning Meeting von dem Entwicklungsteam festgelegt, welche der Aufgaben in dem nächsten Sprint umgesetzt werden können. Es wird dabei nach Priorität vorgegangen. Diese Arbeitspakete werden in dem Sprint-Backlog für jeden Sprint festgehalten. Während des Sprints setzen die Entwickler die Aufgaben im Sprint-Backlog um, während der Productowner den Backlog bearbeitet[133].

Die Verwaltung der Anforderungen im Backlog geschieht in Form von sog. Userstorys. Diese Userstorys sollten die gewünschte Funktionalität, die Rolle des Anwenders, und den Geschäftswert dieser Funktionalität beinhalten. Daneben wird der Aufwand für die Realisierung des Features geschätzt[134]. Der Aufwand wird nicht als absolute Zahl geschätzt, sondern als Aufwand relativ zu anderen Stories geschätzt. Hat Story A eine Storypointanzahl von 1 und B eine Anzahl von 5, ist B mit fünfmal so viel Aufwand verbunden. In einigen Projekten werden Storypoints im gleichen Sinne auch zur Schätzung der Komplexität herangezogen. Von den meisten Scrum-Teams wird eine feste Zahlenfolge zur Schätzung der Aufwände verwendet. Dabei ist die Fibonacci-Folge[135] besonders beliebt.

3.2.2 Aufwandsabschätzungen mit Planning Poker

Eine weit verbreitete Technik zum Schätzen der Userstorys ist das Planungspoker. Zur Schätzung einer Story gibt jedes Teammitglied gleichzeitig und verdeckt eine individuelle Schätzung zu der Story ab. In einem zweiten Schritt wird durch eine Diskussion eine Einigung über das Ergebnis erzielt. Die so erzielten Ergebnisse sollten innerhalb eines Projektes konsequent sein. Auch innerhalb eines Teams über Projekte hinweg können oft konsequente Ergebnisse erzielt werden. Diese Eigenheit wird auch in dieser Arbeit berücksichtigt. So werden die Messwerte der Storypoints auf den Kontext der einzelnen Projekte normalisiert[136].

[133] Vgl. Schwaber/Beedle 2002 und Schwaber 2004
[134] Vgl. Cohn 2004
[135] TODO ERKLÄREN
[136] Vgl. Cohn/Martin 2006 und Dalton 2019, S. 203

4 Forschungsaufbau der Fallstudie

In dieser Arbeit soll eine Fallstudie durchgeführt werden. Entsprechend der Fallstudienmethodik wird in diesem Kapitel der Forschungsaufbau zunächst in einem Forschungsprotokoll festgehalten[137]. Wichtige Komponenten sind dabei zunächst eine Forschungsfrage, welche die Problemstellung und die Ziele des Handelns aufzeigt. Dann wird eine Hypothese über die Antwort aufgestellt. Es werden die einzelnen Fälle bzw. die Analyseeinheiten differenziert, ausgewählt und vorgestellt. Die Daten aus diesen Analyseeinheiten werden in Verbindung zu den Hypothesen gesetzt und es werden Kriterien zur Interpretation der Daten aufgestellt[138].

4.1 Forschungsfrage

Eine klare Forschungsfrage ist essenziell für die Fallstudienforschung, um die Problemstellung und Ziele des Handels aufzuzeigen[139]. In der Fallstudienforschung können Forschungsfragen explorativ, deskriptiv und explanativ sowie eine Kombination aller drei Arten sein[140].

In dieser Arbeit soll das Thema der Softwarekomplexitätsmetriken deskriptiv (Frage nach dem Wie) behandelt werden. Es soll beschrieben werden, wie sich Komplexitätsmetriken im Vergleich zu Aufwandsabschätzungen verhalten.

4.2 Hypothesen

Vor der Beantwortung der Forschungsfrage werden zunächst Hypothesen über mögliche Antworten aufgestellt[141]. Das Aufstellen von Hypothesen dient der Identifikation der weiteren Forschungsrichtung[142].

In dieser Arbeit wird die Hypothese verfolgt, dass eine eingeschränkte Korrelation zwischen Softwarekomplexitätsmetriken und Aufwandsabschätzungen nachgewiesen werden kann. Es ist zu erwarten, dass Störfaktoren diese Korrelation beeinflussen.

[137] Vgl. Göthlich 2003, S. 8 und Mills/Durepos/Wiebe 2010, S. 69f
[138] Vgl. Mills/Durepos/Wiebe 2010, S. 69f
[139] Vgl. Dubé/Paré 2003, S. 605, Mills/Durepos/Wiebe 2010, S. 70 und Dubé/Paré 2003, S. 607
[140] Vgl. Dubé/Paré 2003, S. 605
[141] Vgl. Göthlich 2003, S. 8
[142] Vgl. Dubé/Paré 2003, S. 607 und Mills/Durepos/Wiebe 2010, S. 70

4.3 Analyseeinheit

Die Hypothese soll durch die Betrachtung einer Reihe von Fällen (Cases) als Untersuchungseinheiten validiert oder widerlegt werden.

Im allgemein anerkannten Vorgehen zur Fallstudienforschung sei an dieser Stelle eine klare Beschreibung der Fälle von Bedeutung[143].

Auch an die Auswahl der Fälle bestünden Anforderungen. Die Fälle müssen im Zusammenhang zu der Forschungsfrage gewählt werden, können in diesem Rahmen aber durchaus willkürlich ausgewählt werden[144]. Eine willkürliche Auswahl der Fälle ist gerade deswegen wichtig, da insbesondere Extremfälle die gesamte Bandbreite der möglichen Analyseergebnisse abdecken können. Die willkürliche Auswahl muss jedoch keinem Zufallsprinzip folgen[145]. Als Grundvoraussetzung zur Auswahl der Fälle lässt sich sagen, dass ein möglichst umfassender Zugang zu Dokumenten und Artefakten des Falles gegeben sein sollte[146]. Für eine möglichst valide Endaussage sei eine Anzahl von vier bis zehn Fällen anzustreben[147].

In dieser Arbeit soll die Entwicklung von Softwarekomplexitätsmetriken betrachtet werden. Also liegt es auf der Hand, die Quelltexte von Softwareprojekten als Analyseeinheiten zu betrachten. Die Komplexitätsmetriken sollen mit insgesamt sechs Softwareprojekten validiert werden. Bei fünf Projekten handelt es sich um Projekte, die unternehmensintern im Kundenauftrag entwickelt werden bzw. wurden. Bei einem weiteren Projekt handelt es sich um eine Softwarelösung, die bereits von über 100 Tausend Organisationen verwendet wird[148]. Es lässt sich also sagen, dass der wirtschaftliche bzw. praktische Bezug aller Projekte sichergestellt ist. Aufgrund der Positionierung der ersten fünf Projekte als unternehmensinterne Entwicklungen ist hier des weiteren der weitgehend unlimitierte und vor allem unverfälschte Zugang zu Daten der Projekte sichergestellt. Auch bei dem externen Projekt besteht ein hinreichender Datenzugriff. Somit sind alle zuvor aufgezeigten Voraussetzungen zur Auswahl der Fälle einer Fallstudie bei den hier behandelten sechs Fällen bedient.

Im Sinne einer klaren Beschreibung der Fälle sollen einige Metadaten zu den Fällen gesammelt werden. Die Art der abgefragten Informationen richtet sich nach einer Arbeit von Sato et al (Sato et al 2006:49f) und mehreren Befragungen von Experten in dem Unternehmen. Insgesamt konnten so neun Datenpunkte identifiziert werden.

- Der *Name* des Projektes: Hier musste teilweise aus Gründen der Datensicherheit ein Pseudonym verwendet werden.

[143] Vgl. Dubé/Paré 2003, S. 610
[144] Vgl Mills/Durepos/Wiebe 2010, S. 72f und Yin 2014, S. 72
[145] Vgl. Göthlich 2003, S. 12ff
[146] Vgl. Mills/Durepos/Wiebe 2010, S. 68
[147] Vgl. Göthlich 2003, S. 9 und Dubé/Paré 2003, S. 609
[148] Vgl. *About GitLab | GitLab* 2022

- Eine *Beschreibung*: Diese Beschreibung sollte den Verwendungszweck und eine grobe betriebliche Motivation des Projektes beinhalten.
- *Entwicklungsmethode*: Ob das Projekt agil nach Scrum oder einer anderen Entwicklungsmethode entwickelt wird, ist ebenfalls relevant für die Arbeit.
- *Programmiersprache*: Welche Programmiersprachen in dem Projekt hauptsächlich verwendet werden.
- *Offshore oder Onshore*: Ob die Entwicklung des Projektes in ein anderes Land (offshore) verlegt wurde.
- *Team Lokation*: Der Hauptsitz des Entwicklungsteams.
- *Kunden Lokation*: Der Hauptsitz des Kunden.
- *Erfahrung der Entwickler*: Wie viel Berufserfahrung die Entwickler aufweisen.
- *Erfahrung der Entwickler mit dem Projekt*: Wie viel Erfahrung die Entwickler bereits in dem speziellen Projekt haben.

Die Datenpunkte Name, Beschreibung, Entwicklungsmethode, Sprache, Team Lokation und Kunden Lokation wurden sowohl von Sato et al 2006 und von Experten innerhalb des Unternehmens als wichtig erachtet. Die Relevanz der weiteren Punkte begründet sich allein aus unternehmensinternen Expertenmeinungen.

4.4 Datensammlung

Zu den ausgewählten Quellen werden nun Daten erfasst. Das Ziel der Datenerfassung ist das Schaffen einer belastbaren Basis, auf der eine spätere Belegung oder Widerlegung der Hypothese geschehen kann. Für die Zuverlässigkeit und Validität der getätigten Aussagen sei eine genaue Beschreibung der Datenquellen unerlässlich[149][150]. Die Auswahl und Beschreibung der Datenquellen wird in verschiedenster Literatur umfassend erläutert[151].

Für die Auswahl der Datenquellen kommen verschiedene Quellenformate in Frage: Im Generellen seien Dokumente jeglicher Art als Quelle zulässig[152]. Auch Archivdaten spielen eine wichtige Rolle. So wurde in einer Metastudie ermittelt, dass in ca. 64% aller beobachteten Fallstudien Archivdaten als Quelle eingesetzt wurden[153]. Hier seien insbesondere Zeitreihendaten von Interesse[154]. Weiter seien Interviews, Beobachtungen und Befragungen zulässige Datenquellen[155].

[149] Vgl. Dubé/Paré 2003, S. 612
[150] Vgl. Dubé/Paré 2003, S. 614
[151] Vgl. Girtler 2001, S. 560-569
[152] Vgl. Göthlich 2003, S.10
[153] Vgl. Dubé/Paré 2003, S. 614
[154] Vgl. Benbasat/Goldstein/Mead 1987, Eisenhardt 1989 und Dubé/Paré 2003, S. 612
[155] Vgl. Benbasat/Goldstein/Mead 1987 und Dubé/Paré 2003, S. 612

Auch sämtliche andere Artefakte der Fälle, sowohl in physischer als auch in digitaler Form können zu der Untersuchung hinzugezogen werden[156]

Zu diesen Primärdaten könnten auch noch Sekundärdaten, wie z.B. Interpretationen, Zusammenfassungen, Textanalysen, Statistiken und Berichte hinzugezogen werden[157].

Für eine möglichst valide Endaussage sei eine Kombination der Quellen besonders sinnvoll[158]. Diese Verwendung mehrerer Quellen und auch mehrerer Forschungsmethoden kann als methodologische Triangulation verstanden werden[159].

Bei allen Quellen sei es besonders wichtig, Daten ähnlich wie nach den „Grundsätzen ordnungsgemäßer Buchführung“ in einer geordneten Form lückenlos und unverfälscht aufzubewahren, sodass eine spätere Prüfung der Schlussfolgerungen möglich sei[160].

Im Sinne dieser Empfehlungen zur Auswahl der Quellen wird auch die Auswahl der Quellen in dieser Arbeit geplant. Es werden zwei Primärquellen berücksichtigt. Zum einen werden Dokumente aus der Projektmanagementsoftware der Projekte bezogen. Zum anderen werden Quelltextartifakte aus der Versionsverwaltung analysiert. Zusätzlich zu diesen Primärquellen wird in jedem Projekt ein Abschlussinterview mit einer strukturierten Befragung durchgeführt. Diese Abschlussinterviews sollen die Interpretation der Primärdaten stützten und die ordnungsgemäße Sammlung dieser validieren.

4.5 Verbindung von Daten und Hypothesen

Die eigentliche Beantwortung der Problemstellung bedarf einer Ausrichtung der Daten der Analyseeinheiten (Punkt 3 Units of analysis) als Reflektion der initialen Hypothese[161]. Diese Reflektion bzw. Verbindung kann mit verschiedenen Analyseverfahren hergestellt werden. Dabei stehen unter anderem Mustervergleiche, Erklärungsgebilde, Zeitserienanalysen, logische Modelle und fallübergreifende Synthesen zur Verfügung[162]. Wenn bekannt ist, dass einige oder alle Hypothesen eine Entwicklung über Zeit betrachten, könne zum Beispiel eine Zeitserienbetrachtung sinnvoll sein[163].

In dieser Arbeit wird die Komplexität der Software in Projekten betrachtet (siehe Unit of Analysis). Dabei sollen die Komplexitätsmetriken mit Aufwandsabschätzungen verglichen werden. Dieser Vergleich bzw. diese Verbindung ist in Abbildung .. dargestellt und wird im Folgenden erläutert.

[156] Vgl. Göthlich 2003, S. 10 und Dubé/Paré 2003, S. 612
[157] Vgl. Göthlich 2003, S. 11
[158] Vgl. Göthlich 2003, S. 10
[159] Vgl. Göthlich 2003, S. 10
[160] Vgl. Göthlich 2003, S. 11
[161] Vgl. Yin 2014, S. 78 und Göthlich 2003, S. 11
[162] Vgl. Yin 2014, S. 78 und Göthlich 2003, S. 11
[163] Vgl. Yin 2014, S. 78 und 225

GRAFIK

Für jedes Softwareprojekt sind Daten zur Codekomplexität und zu den Aufwandsabschätzungen vorhanden.

Die Codekomplexität kann aus dem Sourcecode der Software berechnet werden (siehe Analysesoftware). Der Sourcecode aller Projekte wird jeweils in einer Versionsverwaltungssoftware verwaltet. Mit einer solchen Versionsverwaltungssoftware lassen sich jegliche Änderungen am Code historisch nachvollziehen. Gleichzeitig kann so auch für jeden Änderungszeitpunkt in retrospektive der Sourcecode rekonstruiert werden. Also lässt sich über die komplette Entwicklungsgeschichte der Software hinweg rekonstruieren, wie der Sourcecode zu dem jeweiligen Zeitpunkt ausgesehen haben muss. Aus der Historie des Sourcecodes lässt sich auch eine Historie der Sourcecodekomplexität berechnen. Dabei wird zu jedem Zeitpunkt in Retrospektive der Sourcecode hinsichtlich seiner Komplexität analysiert. Auf diese Weise lässt sich eine Zeitserie der Komplexität des Sourcecodes über die Entwicklungsgeschichte hinweg rekonstruieren. Auf der linken Seite von Abbildung .. ist beispielhaft eine solche Serie über drei Zeitpunkte hinweg skizziert. In realen Projekten lassen sich zwischen 1000 und 50.000 distinktive Zeitpunkte rekonstruieren. Die Abbildung hier dient beispielhaft dazu, den Anstieg der Komplexität der Software zu veranschaulichen.

Als zweite Vergleichsgröße sollen Aufwandsabschätzungen dienen. Auch hier lässt sich eine Zeitserie aufbauen. Alle Projekte werden nach der agilen Scrum Methodik verwaltet (Verweis zu Kapitel 3). Diese Methodik sieht vor, im Voraus zu der Implementierung eines Softwarefeatures den Umfang bzw. Implementierungsaufwand dieses abzuschätzen. Diese Abschätzungen werden numerisch in Form von sog. Storypoints abgegeben und in einer Projektmanagementsoftware (wie z.B. Jira vermerkt). Auch wird zusammen mit der Aufwandsabschätzung unter anderem der Anfangs- und der Endzeitpunkt der Implementierung vermerkt. Diese Daten sind bei allen betrachteten Projekten über die komplette Entwicklungsdauer hinweg verfügbar. Es lässt sich also rekonstruieren, zu welchem Zeitpunkt welcher geschätzte Aufwand in die Software geflossen ist. Die Kombination der Komplexitätsabschätzung zusammen mit ihrem Zeitpunkt ergibt ebenfalls eine Zeitreihe. Diese ist in Abbildung ..TODO beispielhaft auf der linken Seite skizziert. Die Entwicklung von Features in einer Software erfolgt konsekutiv und konstruktiv. Das heißt, wird an Zeitpunkt 1 Feature 1 und an Zeitpunkt 2 Feature 2 entwickelt, besteht die Software zu Zeitpunkt 2 aus den Features 1 und 2. Dieser Sachzusammenhang wird ebenfalls in Abbildung .. skizziert. Der insgesamte, abgeschätzte Funktionsumfang der Software zu einem Zeitpunkt X besteht also aus allen Features, die zum Zeitpunkt X entwickelt wurden, addiert zu den summierten Features, welche bereits vor Zeitpunkt X entwickelt wurden. Die Aufwandsabschätzungen der Features (Stories) sind zu den distinktiven Zeitpunkten in Form von Storypoints verfügbar. Also lässt sich durch die Kumulierung aller Aufwandsabschätzungen (Storypoints) bis zu diesem Zeitpunkt der abgeschätzte Funktionsumfang der Software zu diesem Zeitpunkt ableiten. Dieser Sachzusammenhang ist auf der rechten Seite von Abbildung .. dargestellt.

Zusammengefasst konnten bis hierhin zwei Zeitreihen konstruiert werden. Einmal auf der linken

Seite von Abbildung .. die berechnete Codekomplexität der Software und dann auf der rechten Seite die abgeschätzte Komplexität der Software. Beide Werte liegen zeitdistinktiv zu einer Vielzahl von Zeitpunkten vor. Betrachtet man nun einen einzelnen Zeitpunkt, sollte laut der anfänglichen Hypothese der abgeschätzte Gesamtaufwand der Software der berechneten Codekomplexität entsprechen. Zur Validierung der Hypothese lässt sich eine Korrelation mathematisch berechnen. Somit ist über die Korrelation der beiden Zeitreihen eine Verbindung der Fälle (Units of Analysis) über die gesammelten Daten mit der anfänglichen Hypothese hergestellt.

4.6 Kriterien zur Interpretation der Daten

Die in 4.5 erläuterte Korrelation der Aufwandsabschätzungen mit den Codekomplexitätsmetriken soll in einem finalen Schritt interpretiert werden. Im Sinne einer typischen Fallstudienforschung soll dabei eine Erklärung für die aufgetretenen Phänomene gefunden werden[164].

In der einschlägigen Literatur über die Durchführung von Fallstudien finden sich verschiedene Techniken zur Interpretation der gesammelten Daten. Darunter sind unter anderen Techniken der statistischen Analyse[165] sowie Interviews[166]. Mit diesen beiden Techniken werden die Analyseergebnisse aus 4.5 zunächst interpretiert und dann validiert.

Im Sinne einer objektiven Vergleichbarkeit erscheint eine mathematische Interpretation der Daten mit Methoden der Statistik sinnvoll. Auch in verwandten Arbeiten wird mit mathematischen Verfahren gearbeitet. Diese Arbeit strebt einen Nachweis einer Korrelation zwischen zwei Zeitreihen an. Demnach erscheint es sinnvoll, Korrelationsmaße für den Vergleich von Zeitreihen anzuwenden. In verwandten Arbeiten kommen unter anderem die Pearson-Produkt-Moment-Korrelation, der Kendall Rangkorrelationskoeffizient sowie der Spearman Rangkorrelationskoeffizient zum Einsatz[167].

Die Pearson-Produkt-Moment Korrelation ist ein parametrisches Verfahren zur Berechnung des bivariaten Zusammenhangs zwischen zwei Größen. Der Korrelationskoeffizient kann Werte zwischen -1 und 1 annehmen. Bei einem Wert von +1 besteht ein vollständig positiver linearer Zusammenhang. Bei einem Wert von -1 ein vollständig negativer. Das Korrelationsmaß wurde erstmals von Sir Francis Galton verwendet und von Karl Pearson zuerst formal-mathematisch begründet[168].

Die Ergebnisse können über ihren Betrag grob eingeordnet werden[169]:

- „schwache Korrelation“ r <= 0.5
- „mittlere Korrelation“ 0.5 <= r <= 0.8

[164] Vgl. Göthlich 2003, S. 11
[165] Vgl . Yin 2014, S. 79 und Göthlich 2003, S. 11
[166] Vgl. Göthlich 2003, S. 11
[167] Vgl. Jay u. a. 2009, S. 140 und Hindle, Abram/Godfrey, M. W./Holt, R. C. 2009
[168] Vgl. Brückler 2018, S. 116
[169] Vgl. Fahrmeir u. a. 2016, S. 130

- „starke Korrelation“ 0.8 <= r.

Ein weiterer Korrelationskoeffizient ist der Kendall Rangkorrelationskoeffizient. Zur Berechnung des Koeffizienten der die Ordnungsrelationen aller möglicher Paare der beobachteten Merkmalswerte für die zwei Merkmale verglichen. Im Vergleich zu Pearson ist dieser Koeffizient robuster gegenüber Ausreißern und unabhängig von der metrischen Skalierung der Daten[170]. Die Ergebnisse sind jedoch gleich zu interpretieren.

Ein dritter Korrelationskoeffizient ist der Spearman Rangkorrelationskoeffizient. Dieser wird ähnlich wie der Rangkorrelationskoeffizient nach Kendal berechnet, zieht jedoch zusätzlich zu den Unterschieden zwischen den Rängen noch die Differenz zwischen den Rängen hinzu[171]. Auch hier sind die Ergebnisse gleich wie bei den anderen beiden Koeffizienten zu interpretieren.

Zur Validierung dieser Interpretation wird zusätzlich in jedem der Projekte ein Interview mit den Key-Stakeholdern durchgeführt. Das Interview verfolgt dabei drei Ziele. Zunächst werden generelle Informationen zu dem Projekt aufgenommen. Dann werden die Stakeholder befragt, ob ihnen offensichtliche Messfehler in den Projekten auffallen. In einem letzten Schritt werden in einer offenen Diskussion interessante Punkte in den Arbeitsergebnissen identifiziert. Diese Interessenspunkte werden dann versucht zu erklären.

Während bei vielen Forschungsmethoden die genaue Transkription und Analyse von wenig strukturierten Interviews (z.B. durch die Inhaltsanalyse nach Mayring) im Fokus steht, wird bei dieser Arbeit ein strukturiertes Vorgehen bei der Durchführung angestrebt, um die Interviews so durch logische Schlüsse analysieren zu können. Dieses Vorgehen wird auch von verbreiteter Literatur zum generellen Vorgehen bei der Fallstudienforschung unterstützt[172]. Dabei birge eine zu detailreiche Analyse der Interviews die Gefahr der Überinterpretation. Wichtiger sei es, die Ergebnisse der Interviews kritisch zu reflektieren[173]. Für ein möglichst strukturiertes Vorgehen in den Interviews wurde vorab mit mehreren Experten aus dem Feld der Softwareentwicklung ein Leitfaden in Form eines Fragebogens erstellt. Dieser kann Anhang .. entnommen werden.

Die so erlangten Ergebnisse der einzelnen Fälle sollen auch über die Fälle hinweg verglichen und interpretiert werden. Kommen bei einer Fallstudie mehrere Fälle zu dem gleichen Ergebnis spricht man von einer Replikation[174]. Wenn bei allen Fällen eine Korrelation von Aufwandsabschätzungen und Komplexitätsmaßen festgestellt werden kann, ließe dies auf eine allgemeine Regel schließen. Aber auch eine sog. theoretische Replikation ist vorstellbar. Dabei kommen verschiedene Fallstudien zu verschiedenen Ergebnissen, welche aber theorieseitig erklärbar sind[175].

[170] Vgl. Fahrmeir u. a. 2016, S. 137ff
[171] Vgl. Fahrmeir u. a. 2016, S. 133f
[172] Göthlich 2003, S. 12
[173] Göthlich 2003, S. 12
[174] Göthlich 2003, Vgl. [S. 11
[175] Göthlich 2003, S. 11

4.7 Implementierung einer Untersuchungssoftware

Für die Untersuchung der Softwarekomplexitätsmetriken müssen einige komplexe Berechnungen durchgeführt werden. Die Datenbasis dieser Untersuchung umfasst mehrere zehntausend Datenpunkte. Eine manuelle Verarbeitung dieser Daten erscheint als wenig ökonomisch. Also bedarf es einer automatisierten Verarbeitung der Daten. Die Verarbeitung der Daten soll mit einer Software erfolgen. In einer umfangreichen Suche konnte lediglich die Software SonarQube einen wesentlichen Anteil der Anforderungen an ein solches Produkt abdecken. Bei dem Ansetzen einer ersten SonarQube Instanz stellte sich jedoch heraus, dass die Import- und Exportfunktionen von SonarQube unzureichend für die hier angestrebte Untersuchung sind. Da außer SonarQube keine passende Software gefunden werden konnte, liegt es auf der Hand, eine eigene Untersuchungssoftware zu implementieren.

4.7.1 Anforderungen an die Untersuchungssoftware

Die Berechnung der Korrelation von Codekomplexitätsmetriken und Komplexitätsabschätzungen stellt eine Reihe an Anforderungen an die Untersuchungssoftware. Zunächst werden die Anforderungen in einem Überblick zusammengefasst und dann im späteren Verlauf dieses Kapitels genauer betrachtet.

Als Eingabe werden zwei Datenströme vorgesehen. Auf der einen Seite die Entwicklungshistorie der Software und auf der anderen Seite die historischen Komplexitätsabschätzungen für die Features der Software. Besonders zu bemerken ist, dass beide Datenströme irreguläre Zeitintervalle und unterschiedliche Wertebereiche haben.

Funktionale Anforderungen:

- Verarbeitung passender Eingabedatenströme
  - Als ersten Eingabeparameter sind die Komplexitätsabschätzungen vorgesehen. Wie in 4.4 beschrieben, liegen bei allen behandelten Projekten für jedes entwickelte Feature Komplexitätsabschätzungen in einer Projektmanagementsoftware vor. Aus dieser Software können die Schätzungen in Tabellenform als CSV-Datei exportiert werden. Der Export soll unverarbeitet als ersten Eingabeparameter in die Untersuchungssoftware eingelesen werden. Dabei muss diese Tabelle zwei Spalten beinhalten. Ein Datensatz in dieser Tabelle muss jeweils einen Zeitstempel mit dem Entwicklungszeitpunkt des Features, sowie die Komplexitätsabschätzung als Anzahl von Storypoints beinhalten.
  - Als zweiten Eingabeparameter soll die Versionshistorie der Software dienen. Bei allen zu untersuchenden Projekten liegt die Versionshistorie in Form eines sog. Repositories der Versionsverwaltungssoftware Git[176] vor. Diese Versionshistoriendaten sollen von der Untersuchungssoftware verarbeitet werden können.

[176] TODO Git erklären

- Generierung zielgerichteter Ausgangsdatenströme
  - Zunächst wird als erster Ausgangsdatenstrom vorgesehen, dass in einem regulären Zeitintervall von einer Stunde über die komplette Entwicklungshistorie der Software hinweg jeweils die kumulierten Storypoints, sowie der jeweilige Stand der Komplexitätsmetriken vorliegen. Ist zu dem spezifischen Zeitpunkt in dem Zeitintervall kein Wert vorhanden, soll dieser aus den Nachbarwerten linear interpoliert werden. Als Ausgabeformat wird eine Tabelle in Form einer Comma-separated values (CSV) Datei vorgesehen:

TODO!TABLE

- Als zweiter Ausgabeparameter wird ein Graph des Verlaufes aller Maßzahlen im Portable Network Graphik (PNG)- und Portable Document Format (PDF)-Format vorgesehen.
- In einem dritten Ausgabeparameter soll die Korrelation der Maßzahlen mit den Storypoints mit verschiedenen Korrelationsmaßen dargestellt werden. Als Korrelationsmaße ist die Pearson-Produkt-Moment-Korrelation, die Kendall Rangkorrelation und die Spearman Rangkorrelation vorgesehen. In dem dritten Ausgabeparameter sollen diese Koeffizienten als Tabelle im CSV-Format für jedes Komplexitätsmaß angegeben werden.
- Als vierten Ausgabeparameter ist eine Heatmap[177] mit den Korrelationskoeffizienten auf der x-Achse und den Komplexitätsmetriken auf der y-Achse vorgesehen.
- Für eine möglichst breite Anwendbarkeit soll die Software unter Unix-basierten Betriebssystemen, wie Linux und Mac ausführbar sein. Außerdem können so eine Vielzahl von Open-Source Bibliotheken angebunden werden.

Nicht-funktionale Anforderungen

- Der Fokus dieser Arbeit liegt nicht auf der Implementierung der Untersuchungssoftware, sondern auf den mit ihr erzielten Ergebnissen. Für eine möglichst ökonomische Beantwortung der Forschungsfragen erscheint es also von Interesse, die Untersuchungssoftware mit einem möglichst geringen Implementierungsaufwand umzusetzen.
- In dieser Arbeit sollen einer Vielzahl von teils sehr umfangreichen Projekten analysiert werden. Um eine zeitgerechte Abwicklung der Analysen zu gewährleisten, soll der Untersuchungssoftware möglichst effizient mit Rechenressourcen umgehen.

[177] TODO Definition Heatmap

4.7.2 Aufbau der Untersuchungssoftware

Zur Begegnung der zuvor genannten Anforderungen wird eine Untersuchungssoftware skizziert. Die Transformation der Eingabedaten zu den Ausgabedaten bedarf einer Vielzahl von komplexen Operationen. Für eine ökonomische Umsetzung der Software (siehe Anforderung ..) wird angestrebt, einen möglichst großen Anteil des Funktionsumfangs der Software durch eine Wiederverwendung von bereits vorhandenen Softwaremodulen zu realisieren. Im Zuge dessen wird auch eine Implementierung über mehrere Programmiersprachen hinweg in Kauf genommen. Eine Skizze der so erreichten Implementierung wird in Abbildung TODO! dargestellt und im Folgenden erklärt.

TODO!GRAFIK

Der Untersuchungssoftware ist in drei Teile unterteilt. Zunächst liest der Code Parser (Zahl) die Versionshistorie der Software ein und berechnet für jeden Änderungsstand (Commit) die Codekomplexitätsmetriken der Software. Die so berechneten Metriken werden in einem zweiten Modul (2) mit den Daten aus dem Projektmanagement Tool zusammengefügt, verarbeitet und zu den Ausgabedaten aufbereitet.

Der Code Parser

Der Code Parser ist dafür verantwortlich, die einzelnen Entwicklungsstände der Software aus der Versionshistorie zu extrahieren und für jeden Entwicklungsstand die Komplexitätsmetriken zu berechnen. Als Eingabeparameter ist ein Repository der Versionsverwaltungssoftware Git vorgesehen. Als Ausgabeparameter sind die Softwarekomplexitätsmetriken in Tabellenform als CSV Datei vorgesehen (Tabelle ..). Der Code Parser besteht aus einigen generellen Kontrollstrukturen und speziellen Analysemodulen, welche die Komplexitätsmetriken für verschiedene Programmiersprachen in den Projekten berechnen. Für die Analysemodule wird auf bereits verbreitete Bibliotheken zurückgegriffen, da eine Eigenentwicklung der Module zu aufwändig wäre. Bei der Auswahl der Bibliotheken wurde darauf geachtet, möglichst gut gewartete Software zu verwenden, um so ein möglichst genaues Ergebnis zu erzielen. Eine Tabelle mit allen betrachteten Bibliotheken findet sich in Anhang .. . Für ein noch genaueres Ergebnis werden manche Metriken doppelt berechnet. In dem Zahlenverarbeitungsmodul wird später das jeweils genauste Ergebnis ausgewählt. Der Ablauf des Code Parsers wird im folgenden erläutert:

1. **Commit Getter**: Zunächst werden durch den Commit Getter alle Entwicklungsstände der Software mit ihrem jeweiligen Commit-Hash und dem Zeitstempel des Eincheckzeitpunktes in eine CSV Datei geschrieben

2. **Commit Runner:** Der Commit Runner liest diese CSV-Datei, iteriert über alle Commits und führt für jeden Commit der Software alle Analysemodule aus. Die Analysemodule schreiben jeweils die Ergebnisse der Analyse des jeweiligen Commits in eine CSV-Datei mit dem Commit-Hash als Dateinamen.

a. **Lizard**: Das Python Modul Lizard berechnet für alle Programmiersprachen die zyklomatische Komplexität[178].

b. **Plato**: Das NodeJS Commandline-Tool ES6-Plato[179] kann lediglich JavaScript Quelltexte analysieren. Für die JavaScript Teile des zu untersuchenden Projektes berechnet Plato noch einmal die zyklomatische Komplexität, den Halstead Aufwand und die Anzahl logischer Codezeilen

c. **Indentation Complexity**: Mit dem Rust Commandline-Tool Complexity der Firma thoughtbot, inc[180] wird die Einrückungskomplexität für alle Programmiersprachen berechnet

d. **MulitMetric**: Das MultiMetric Python Modul[181] ist zuletzt in der Lage für sämtliche hier behandelte Programmiersprachen die zyklomatische Komplexität, den Halstead Aufwand und die Anzahl logischer Codezeilen zu berechnen.

3. **Consolidator:** Als letzter Schritt werden im Consolidator die Ergebnisse aller Analysemodule für alle Entwicklungsstände in einer Tabelle konsolidiert und als eine CSV-Datei ausgegeben. Dafür liest der Consolidator zunächst die Commit-Hashes und Zeitstempel aus der, von dem Commit-Getter generierten CSV Datei. Für jeden Commit wird für jedes Analysemodul nach einer Ergebnis-CSV-Datei gesucht. Für jede Komplexitätsmetrik wird eine Summe[182] aller Dateien in dem Analyseergebnis gebildet. Die Summen der Komplexitätsmetriken der einzelnen Entwicklungsstände werden in einer CSV-Datei ausgegeben.

Die Ausgabe des Consolidators ist gleichzeitig auch die Ausgabe des Code Parsers. Als solche wird sie in Tabelle .. beschrieben.

TODO!TABELLE

4.7.3 Zahlenverarbeitungsmodul

Nachdem in dem Code Parser die Komplexitätsberechnungen stattgefunden haben, sollen diese nun in dem Zahlenverarbeitungsmodul zusammen mit den Aufwandsabschätzungen zu dem Analyseergebnis verarbeitet werden.

Das Zahlenverarbeitungsmodul verarbeitet zwei Eingabeparameter. Zunächst wird das Resultat des Code-Parser-Moduls (siehe Tabelle ..) gelesen. Wie in 4.7.1 beschrieben ist als zweiter Eingabeparameter der Export von einer Projektmanagementsoftware vorgesehen. In diesem Export sollte eine CSV-Tabelle mit Datensätzen von den Features der Software enthalten sein. Jeder Datensatz beinhaltet mindestens zwei Datenpunkte, einen Zeitstempel (date) und eine

[178] Vgl. *Lizard Python Module* o.J.
[179] Vgl. *Es6-Plato/Lib at Master · Deedubs/Es6-Plato* 2022
[180] Vgl. *Complexity* 2022
[181] Vgl. Weihman 2021
[182] Vgl. Hoffmann 2013, S. 276

Komplexitätsabschätzung (storypoints). Eine Beispiel-Eingabetabelle kann Tabelle .. entnommen werden.

TODO!TABELLE

Die Operationen innerhalb dieses Moduls befassen sich primär mit der Analyse und Verarbeitung großer Datenmengen. Das Wissenschaftsfeld Data Science behandelt ebenfalls genau diese Operationen. In Data Science ist die Programmiersprache Python weit verbreitet. Also liegt es auf der Hand, auch für dieses Modul die Programmiersprache Python zu verwenden. Für Python existiert eine Vielzahl von Bibliotheken zur Verarbeitung von großen Datenmengen. Für dieses Modul wird primär die Datenanalyse- und Manipulationsbibliothek Pandas verwendet[183]. Zusätzlich wird die Bibliothek Matplotlib zum Erstellen von Graphen zur Hilfe gezogen. In der Analyse der Daten erscheint es zusätzlich sinnvoll, Zwischenergebnisse interaktiv zu verifizieren. So können Analysefehler bereits früh erkannt und behoben werden. Die Software Jupyter ermöglicht genau solch eine interaktive Programmausführung und wird somit zur Ausführung des Moduls herangezogen.

Der Aufbau und die Implementierung des Zahlenverarbeitungsmoduls werden nun erläutert. Das Modul lässt sich in vier Teile aufteilen:

1. Verarbeiten der Codekomplexitätsmessungen zu einer Zeitreihe
2. Verarbeiten der Aufwandsabschätzungen zu einer Zeitreihe
3. Zusammenführen beider Zeitreihen
4. Analysieren der Zeitreihen

Im ersten Schritt zur Verarbeitung der Codekomplexitätsmessungen werden diese zunächst als Pandas Dataframe in das Modul geladen (Zeile 1). Bis auf den Zeitstempel können alle Datentypen automatisch erkannt werden. Der Datentyp des Zeitstempels wird manuell gesetzt (Zeile 2)

```
1 codeAnalysisInput = pd.read_csv(CODE_CSV_PATH, parse_dates=['
    timestamp'])
2 codeAnalysisInput['timestamp'] = pd.to_datetime(codeAnalysisInput.
    timestamp, utc=True)
```

Dann wird der Zeitstempel als Zeitindex gesetzt (Zelle 5 Zeile 1-3)

```
1 codeAnalysisInput = codeAnalysisInput.set_index(pd.DatetimeIndex(
    codeAnalysisInput['timestamp']))
2 codeAnalysisInput.drop('timestamp', axis=1)
3 codeAnalysisInput = codeAnalysisInput.tz_convert('UTC')
```

[183] Vgl. *Pandas - Python Data Analysis Library* 2022

Als zweiter Schritt ist eine Untersuchung der gelesenen Daten vorgesehen. Dabei wird zunächst untersucht, welche Ergebnisse in der Eingabe vorhanden sind (Zelle 7 Zeile 1-4).

```
1 measures = []
2 for seriesName in codeAnalysisInput.columns:
3     if seriesName != 'commit' and seriesName != 'timestamp':
4         measures.append(seriesName)
5
6 measures
```

Weiter wird eine Übersicht erstellt, zu welchen dieser Ergebnisse valide Werte vorliegen (Zelle 8 Zeile 1-17). Dabei werden alle Ergebnisse aussortiert, bei denen mehr als zehn Prozent der Messwerte ungültig sind.

```
1 resultsInfo = {}
2
3 for seriesName in measures:
4     series = codeAnalysisInput[seriesName]
5     numberOfNaN = series.isna().sum()
6     proportionOfInvalid = numberOfNaN / len(series)
7
8     if proportionOfInvalid > 0.1:
9         valid = False
10    else:
11        valid = True
12
13    resultsInfo[seriesName] = {
14        'numberOfNaN': numberOfNaN,
15        'proportionOfInvalid': proportionOfInvalid,
16        'valid': valid
17    }
```

Nun wird für jede zu berechnende Komplexitätsmetrik (Anzahl logischer Codezeilen, zyklomatische Komplexität, Halstead Aufwand und Einrückungskomplexität) das beste Ergebnis aus den vorliegenden Ergebnissen der Analysemodule ausgewählt (Zelle 9). Für diese Auswahl wurde zuvor in dem Zahlenverarbeitungsmodul ein Zuweisungsobjekt hinterlegt (Zelle 2). In diesem Zuweisungsobjekt wird für jede Maßzahl eine Präferenzliste der möglichen Ergebnisse der Analysemodule hinterlegt. In dieser Zelle wird nun über die zu berechnenden Komplexitätsmetriken iteriert und für jede Metrik das erste valide Analyseergebnis aus der Präferenzenliste ausgewählt. Die Endauswahl wird in einer Variable abgespeichert.

```
1 measuresToExamine = {}
2
3 for measure in MEASURE_MAPPING:
```

```
4       for parserMeasure in MEASURE_MAPPING[measure]:
5           if resultsInfo[parserMeasure]['valid']:
6               measuresToExamine[measure] = parserMeasure
7               break
```

Anhand dieser Zuweisungstabelle wird nun ein Dataframe mit den besten Analyseergebnissen für jede Metrik aufgebaut. Für alle Datenpunkte, bei denen an dieser Stelle noch kein Wert vorhanden ist, muss davon ausgegangen werden, dass kein Wert ermittelt werden konnte. Diese werden mit 0 aufgefüllt (Zelle 10, Zeile 6)

```
1 dataFrameContents = {}
2
3 for measure in measuresToExamine:
4     dataFrameContents[measure] = codeAnalysisInput[measuresToExamine[
          measure]]
5
6 measureResults = pd.DataFrame(data=dataFrameContents)
7 measureResults.fillna(0, inplace=True)
```

Bis zu diesem Punkt konnte nun eine Tabelle mit den Datensätzen der Komplexitätsmessungen aufgebaut werden. Ein Beispiel ist in Tabelle .. aufgeführt. Jeder Datensatz beinhaltet einmal den Zeitstempel der Komplexitätsmessung (timestamp), die Anzahl logischer Codezeilen (logicalLinesOfCode), die zyklomatische Komplexität (cyclomaticComplexity), den Aufwand nach Halstead (halsteadEffort) und die Einrückungskomplexität (indendationComplexity).

TODO!TABELLE

Als nächster Schritt werden die Daten in einen einheitlichen Wertebereich normalisiert (Zelle 11). Dabei werden die Daten so linear verschoben und skaliert, dass jeweils der erste Wert jeder Zeitreihe 0 ist und der letzte Wert 1 ist.

```
1 measureResultsNormalized = measureResults.copy()
2
3 for measure in measureResults.columns:
4     firstNumber = measureResults[measure].iloc[0]
5     lastNumber = measureResults[measure].iloc[-1]
6     measureResultsNormalized[measure] = (measureResultsNormalized[
          measure] - firstNumber) / (lastNumber - firstNumber)
```

Zusätzlich soll nun auch das Zeitintervall normalisiert bzw. regularisiert werden. Dabei werden die Daten zunächst auf ein geringes Intervall von 60 Sekunden abgetastet. Die fehlenden Werte werden linear interpoliert. In einem durchschnittlichen Projekt werden so ca. 2 Millionen Datensätze generiert. Diese Anzahl von Datensätzen ist jedoch zu groß, um sie weiter zu verarbeiten. Für

die weitere Verarbeitung wird die Zeitreihe erneut auf ein Intervall von einer Stunde abgetastet, was die Anzahl an Datensätzen auf ca. 34 tausend reduziert.

```
1 measureResultsResampled = measureResultsNormalized.resample('60S').
      mean()
2 measureResultsInterpolated = measureResultsResampled.interpolate(
      method='linear')
3 measureResultsResampledToHours = measureResultsInterpolated.resample(
      '1H').mean()
```

Bis zu diesem Punkt wurde für jede Komplexitätsmetrik eine passende Messreihe ausgewählt und die Werte auf ein einheitliches Zeitintervall, sowie einen einheitlichen Wertebereich normalisiert. Diese Daten können in dieser Form zusammen mit den Aufwandsabschätzungen weiterverarbeitet werden.

Als zweiter Arbeitsschritt befasst sich das Zahlenverarbeitungsmodul mit der Verarbeitung der Aufwandsabschätzungen. Diese Daten werden ebenfalls zunächst in einen Pandas Dataframe geladen und bereinigt (Zelle 13).

```
1 jiraImport = pd.read_csv(PM_CSV_PATH, parse_dates=[PM_DATE_COLUMN])
2 jiraImportFiltered = jiraImport[jiraImport[PM_DATE_COLUMN].notnull()]
```

Weiter werden die Aufwandsabschätzungen nach ihrem Zeitpunkt aufsteigend sortiert (Zelle 14).

```
1 jiraImportSorted = jiraImportFiltered.sort_values(by=PM_DATE_COLUMN)
```

Weiter werden die relevanten Spalten aus den Daten ausgewählt (Zeile 1), der Index als Zeitstempel in der Coordinated Universal Time (UTC) Zeitzone gesetzt (Zeile 2 und 3), die Spalten mit einheitlichen Namen versehen und dann das resultierende Datenset noch einmal nach Datum aufsteigend sortiert (Zeile 5).

```
1 pmData = jiraImportPreselected[[PM_STORYPOINT_COLUMN]].copy()
2 pmData = pmData.set_index(pd.DatetimeIndex(jiraImportPreselected[
      PM_DATE_COLUMN]))
3 pmData = pmData.tz_convert('UTC')
4 pmData = pmData.rename(columns={PM_STORYPOINT_COLUMN: 'storypoints',
      PM_ISSUE_COLUMN: 'issue'})
5 pmData = pmData.sort_index(ascending=True)
```

Nun werden die Storypoints wie in 4.5 beschrieben kumuliert (Zelle 15). Hierzu bietet Pandas die Funktion cumsum an[184].

[184] Vgl. *Pandas.DataFrame.Cumsum — Pandas 1.4.2 Documentation* 2022

```
pmDataAccumulated = pmData.copy()
pmDataAccumulated['storypoints'] = pmDataAccumulated['storypoints'].
    cumsum()
```

Dann wird in dem gleichen Verfahren wie bei den Codekomplexitätsmetriken der Wertebereich der Aufwandsabschätzungen auf einen Bereich zwischen 0 und 1 normalisiert (Zelle 16).

```
pmDataNormalized = pmDataAccumulated.copy()
lastValue = pmDataNormalized['storypoints'].iloc[-1]
pmDataNormalized['storypoints'] = pmDataNormalized['storypoints'] /
    lastValue
```

Auch das Zeitintervall der Messdaten wird zu einem regulären Zeitintervall von einer Stunde vereinheitlicht (Zelle 17).

```
pmDataResampled = pmDataNormalized.resample('60S').mean()
pmDataInterpolated = pmDataResampled.interpolate(method='linear')
pmDataResampledToHours = pmDataInterpolated.resample('1H').mean()
```

Das resultierende Datenset hat nun also genau wie die Codemetriken einen Wertebereich zwischen null und eins und eine Abtastrate von 60 Sekunden. Die beiden Zeitreihen sind nun also sowohl in ihrer Skalierung als auch in ihrer Abtastrate identisch.

Als dritter Schritt des Zahlenverarbeitungsmoduls können die Zeitreihen der Codekomplexitätsmessungen und der Aufwandsabschätzungen nun zusammengeführt werden. Das geschieht mit der Merge-Funktion von Pandas[185]. Die Merge-Funktion stellt eine neue Tabelle mit den Werten der Codekomplexitätsmessungen und der Aufwandsabschätzungen auf. Durch den „how=inner“ Parameter wird festgelegt, dass ein Inner-Merge durchgeführt wird. Bei einem Inner-Merge werden nur Zeitpunkte berücksichtigt, an denen in beiden Zeitreihen ein Wert verhanden ist.

```
dataMerged = pd.merge(pmDataResampledToHours,
    measureResultsResampledToHours, how='inner', left_index=True,
    right_index=True)
```

Die zusammengefügten Daten werden nun noch einmal zusammen normalisiert (Zelle 22)

```
dataMergedNormalized = dataMerged.copy()

for column in dataMergedNormalized.columns:
    firstNumber = dataMergedNormalized[column].iloc[0]
    lastNumber = dataMergedNormalized[column].iloc[-1]
    dataMergedNormalized[column] = (dataMergedNormalized[column] -
        firstNumber) / (lastNumber - firstNumber)
```

[185] Vgl. *Pandas.DataFrame.Merge — Pandas 1.4.2 Documentation* 2022

Die normalisierten Daten werden als erster Ausgabeparameter der Analysesoftware (siehe 4.7.1) als CSV-Datei exportiert.

Weiter sollen die Daten nun analysiert werden. Dazu wird zunächst mit der Pandas Mean-Funktion[186] eine Zeitreihe des arithmetischen Mittels aller Komplexitätsmetriken berechnet (Zelle 24).

```
1 mean = finalData[['logicalLinesOfCode', 'cyclomaticComplexity', '
    halsteadEffort', 'indendationComplexity']].mean(axis=1)
2
3 finalDataDescribed = finalData.copy()
4 finalDataDescribed['mean'] = mean
```

Nun wird die Differenz dieses arithmetischen Mittels und den Aufwandsabschätzungen gebildet (Zelle 25). Ist diese Differenz besonders groß, heißt das, dass die Komplexitätsmessungen an dieser Stelle besonders stark von den Aufwandsabschätzungen abweichen.

```
1 difference = finalDataDescribed['mean'] - finalDataDescribed['
    storypoints']
2
3 finalDataDescribed['difference'] = difference
4 finalDataDescribed
```

Aus den bis jetzt berechneten Daten kann der Korrelationsgraph aufgestellt werden. Dieser wird als zweiter Ausgabeparameter (siehe 4.7.1) in Form von einer PDF und einer PNG Datei exportiert. Der Code zur Generierung des Korrelationsgraphen findet sich im Anhang TODO.

Als letzter Schritt können die Korrelationen der Komplexitätsmetriken mit den Aufwandsabschätzungen berechnet werden (Zelle 23). Das ist in dem Pandas Modul mit der Corr-Funktion[187] möglich. Die einzelnen Korrelationskoeffizienten werden dann zu einer Tabelle zusammengefügt (Zeile 13) und als CSV-Datei exportiert. Diese CSV-Datei stellt den dritten Ausgabeparameter der Analysesoftware dar (siehe 4.7.1).

Zur Visualisierung der Korrelationskoeffizienten wird zuletzt noch eine Heatmap als vierter und letzter Ausgabeparameter (siehe 4.7.1) erstellt und im PNG und PDF Format exportiert.

Zusammengefasst konnten mit der hier erläuterten Software die Komplexitäten zu den verschiedenen Entwicklungszeitpunkten der Software berechnet werden und zusammen mit den Aufwandsabschätzungen analysiert werden.

[186] Vgl. *Pandas.DataFrame.Mean — Pandas 1.4.2 Documentation* 2022
[187] Vgl. *Pandas.DataFrame.Corr — Pandas 1.4.2 Documentation* 2022

4.8 Grenzen und Probleme

Die hier vorgestellte Analysesoftware unterliegt einer Reihe von Einschränkungen.

Zunächst ist sie in der Verarbeitung der Eingabeparameter beschränkt. Die Revisionsgeschichte der Repositories muss in der Git Versionsverwaltung vorliegen. Git hat laut dem Black Duck Open Hub Repository bei Open Source Projekten zwar einen Marktanteil von 73%[188], jedoch wäre eine Unterstützung für andere Versionskontrollsysteme auch erstrebenswert. Das Eingabeformat der Aufwandsabschätzungen ist zwar auf eine CSV-Datei beschränkt, jedoch ist CSV ein offener Standard und die meisten Tabellenformate lassen sich in dieses Format konvertieren.

Eine weitere Einschränkung der Analysesoftware liegt in der Verarbeitung der Daten. Durch die Normalisierung aller Zeitreihen gehen die absoluten Werte verloren und die Komplexitätsmetriken können nur in Relation zueinander betrachtet werden. Diese Einschränkung ist notwendig, um die Werte vergleichbar zu machen.

[188] Vgl. *Compare Repositories - Open Hub* 2022

5 Untersuchung der Softwarekomplexitätsmetriken

Mit der in Kapitel 4 vorgestellten Methodik werden nun sechs Softwareprojekte als Fälle der Fallstudie untersucht. Dabei wird jedes Projekt zunächst wie in 4.3 beschrieben vorgestellt. Dann wird die Erhebung der Daten beschrieben. Am Ende der Datenerhebung steht für jedes Projekt ein Diagramm zu Verfügung. Für das erste Projekt wird dieses Vorgehen beispielhaft ausführlich erklärt. Hier wird auch der Korrelationsgraph erläutert. Zur Vermeidung von Doppelungen wird in den weiteren Projekten auf eine umfangreiche Erklärung des Analysevorgehens verzichtet.

5.1 Digital NDA Application

Die „Digital Non-Disclosure Agreement (NDA) Application“ hat das Ziel, das komplette Handling der Verschwiegenheitserklärungen im DIL und bei potenziellen weiteren Kunden zu digitalisieren. Insbesondere soll die bisher für eine Verschwiegenheitserklärung nötige Papierunterschrift durch eine digitale Signatur ersetzt werden. Dazu wird den DIL-Besuchern die Möglichkeit gegeben, ihre Daten schon vorab über eine Webapplikation zu übermitteln, woraufhin die bereits auf den Besucher angepasste Verschwiegenheitserklärung vor Ort auf einem Tablet signiert werden kann[189].

Das Projekt wurde agil von einem onshore Team aus erfahrenen Entwicklern in Deutschland entwickelt. Es wurden hauptsächlich JavaScript (52%) und TypeScript (30%) als Programmiersprachen verwendet. Zusätzlich fielen in der Analyse noch kleinere Programteile in Gherkin (7%), PLpgSQL (5%), sowie HyperText Markup Language (HTML), Shell, SCSS und der Dockerfile Syntax zu jeweils weniger als 5% Anteil auf.

5.1.1 Datenerhebung

Wie in Kapitel 4.4 beschrieben, sollen für die Fallstudie in jedem Projekt zunächst verschiedene Daten erhoben werden. Für die Berechnung der Korrelation zwischen den Softwarekomplexitätsmetriken und den Aufwandsabschätzungen wird für beide Größen jeweils eine Datenquelle benötigt. Die Daten zu den Aufwandsabschätzungen sollen aus der Projektmanagementsoftware des Projektes bezogen werden und die Daten zu den Softwarekomplexitätsmetriken aus der Versionsverwaltung des Quelltextes.

In dem Digital NDA Application Projekt werden die Aufwandsabschätzungen in der Projektmanagementsoftware Jira verwaltet. In der Projektmanagementsoftware finden die Aufwandsabschätzungen entsprechend der Scrum-Methodik (siehe Kapitel 3) im Rahmen von User Storys statt. In diesen User Storys ist jeweils eine Anzahl an Story Points und ein Datum vermerkt.

[189] Vgl. Technologies 2022

TODO Grafik

Wie in Kapitel 4.4 beschrieben, sollen für die Fallstudie in jedem Projekt zunächst verschiedene Daten erhoben werden. Für die Berechnung der Korrelation zwischen den Softwarekomplexitätsmetriken und den Aufwandsabschätzungen wird für beide Größen jeweils eine Datenquelle benötigt. Die Daten zu den Aufwandsabschätzungen sollen aus der Projektmanagementsoftware des Projektes bezogen werden und die Daten zu den Softwarekomplexitätsmetriken aus der Versionsverwaltung des Quelltextes.

In dem Digital NDA Application Projekt werden die Aufwandsabschätzungen in der Projektmanagementsoftware Jira verwaltet. In der Projektmanagementsoftware finden die Aufwandsabschätzungen entsprechend der Scrum-Methodik (siehe Kapitel 3) im Rahmen von User Storys statt. Ein Beispiel einer User Story findet sich in Abbildung TODO . In jeder User Story werden, wie in Kapitel 3 beschrieben wichtige Informationen zu der geplanten Funktion der Anwendung hinterlegt. Für diese Arbeit sind die Felder „Story Points“ (#1) und „Resolved“ unter Dates (#2) von besonderer Relevanz.

TODO Grafik

Aus der Software konnten insgesamt 67 Storys exportiert werden und dienen wie in 4.7.1 beschrieben als Input für die Analysesoftware.

Als zweite Datenquelle wird die Versionshistorie des Quelltextes der DTCNDA Anwendung herangezogen. Diese liegt in Form eines Git Repositories vor. Aus dem Git Repository konnten 1625 Commits geladen werden. Die Versionshistorie des Quellcodes wird ebenfalls als Eingabeparameter für das Analysetool übernommen.

5.1.2 Auswertung

In dieser Arbeit soll die Korrelation zwischen Softwarekomplexitätsmetriken und Aufwandsabschätzungen beschrieben werden. Nachdem in 5.1.1 für beide Größen Daten gesammelt wurden, sollen diese nun wie in 4.5 beschrieben ausgewertet werden. Hierzu werden die Daten in die in 4.7 beschriebe Analysesoftware geladen und mit ihr verarbeitet. Es werden zunächst vom Code Parser die Codekomplexitätsmetriken für die einzelnen Entwicklungsstände der Software berechnet. Dann wird das Ergebnis dieser Berechnungen in das Zahlenverarbeitungsmodul geladen. Dort werden die Codekomplexitätsmetriken zusammen mit den Aufwandsabschätzungen zu einem einheitlichen Format verarbeitet und in Relation zueinander gesetzt. Diese Relation der Größen wird mit dem Graph in Abbildung TODO beschrieben.

TODO Grafik

Abbildung TODO wird im Folgenden als Korrelationsgraph des Projektes bezeichnet und soll nun erläutert werden. Die Abbildung besteht aus zwei Diagrammen, welche übereinander angeordnet sind. In dem oberen Diagram werden alle behandelten Größen in einem gemeinsamen

Wertebereich abgezeichnet. Die Story Points sind mit einer dickeren, schwarzen Linie dargestellt. Die restlichen Linien beschreiben die Codekomplexitätsmetriken. Die einzelnen Farbzuweisungen können der Legende im oberen linken Rand der Abbildung entnommen werden. Das zweite Diagramm in der Abbildung beschreibt die Differenz zwischen den Story Points und dem arithmetischen Mittel aller Codekomplexitätsmetriken. Es soll anzeigen, wo die Werte der beiden Größen besonders stark voneinander abweichen. Der Wertebereich dieses zweiten Diagramms ist analog zu dem Wertebereich des ersten Diagramms. Das Format dieser Abbildung wird über alle folgenden Projekte hinweg konstant gehalten.

In dem Korrelationsgraph sind die einzelnen Zeitreihen über den Projektzeitraum von Oktober 2018 bis November 2019 abgezeichnet. Alle Zeitreihen folgen einer ähnlichen Linie. Diese könnte näherungsweise als logarithmisch beschrieben werden. Nach einer stärkeren Steigung zum Anfang des Projektes flacht die Steigung aller Größen zum Ende des Projektes zunehmend ab. Dieser Verlauf konnte durch eine Verschiebung in dem Ziel der Softwareentwicklung begründet werden. So wurde die Software zum Ende hin immer langsamer weiterentwickelt. In dem Abschlussinterview konnten die sprungartigen Anstiege der Aufwandsabschätzungen (1) durch die Sprint-Intervalle erklärt werden. So werden die Storys vermehrt zum Sprint-Ende als fertig markiert (Analysis Table Line 20). Des Weiteren wurden Ende November 2018 einige große und kurzfristige Schwankungen in den Komplexitätsmetriken identifiziert (2 in der Abbildung). In diesem Zeitraum wurde die Anwendung restrukturiert. Im Rahmen der Restrukturierung wurden regelmäßig große Codeteile entfernt und wieder hinzugefügt. Diese Schwankungen stehen also in keinem Zusammenhang zu dem tatsächlichen Umfang der Anwendung und können ignoriert werden. Von Mitte Dezember 2018 bis Mitte Januar 2019 ist sowohl in den Aufwandsabschätzungen, als auch in den Codekomplexitätsmetriken ein Plateau erkenntlich (Nummer 4). Dieser Stillstand in der Weiterentwicklung der Software konnte auf einen generellen Betriebsschluss aufgrund der Weihnachtszeit zurückgeführt werden. Zum Ende des Projektes sinkt die Entwicklungsgeschwindigkeit (Nummer 3). Hier wurden Entwickler von dem Projekt abgezogen.

Zusammengefasst entsprechen die hier gesammelten Messdaten der Hypothese. Über den Projektverlauf steigen die berechneten Softwarekomplexitätsgrößen im größtenteils gleichen Maße wie die Aufwandsabschätzungen. Abweichungen und Auffälligkeiten konnten mit den Projektbeteiligten erklärt werden. Dementsprechend ist ein hoher Korrelationsgrad zu erwarten.

In einem nächsten Schritt wurden wie in 4.6 und 4.7 beschrieben die Korrelationskoeffizienten berechnet. Diese können Tabelle TODO entnommen werden.

TODO

Für alle Untersuchungsergebnisse wurde ein P-Wert von $< .00001$ ermittelt. Bei einem, für Sozial- und Wirtschaftswissenschaften üblichen Signifikanzniveau von .05 lässt sich also sagen, dass die Ergebnisse signifikant sind.

Für alle Komplexitätsmetriken konnten mit Korrelationskoeffizienten zwischen 0,85 und 0,98 starke Korrelationen ermittelt werden. Mit einem durchschnittlichen Korrelationskoeffizienten von

0.95 ist die Korrelation bei den logischen Codezeilen am stärksten, gefolgt von der Einrückungskomplexität (0,94), der zyklomatischen Komplexität (0,91) und dem Aufwand nach Halstead (0,90).

5.1.3 Fazit

In diesem Fall konnten erfolgreich eine Zeitreihe der Softwarekomplexitätsmaßzahlen und eine der Aufwandsabschätzungen aufgestellt werden. Es wurden wenige Störfaktoren identifiziert. Letztendlich wurde eine zum größten Teil sehr starke, signifikante Korrelation zwischen den Komplexitätsmaßen und den Aufwandsabschätzungen festgestellt. Der Fall der Digital NDA Application spricht also für eine Korrelation zwischen den Codekomplexitätsmetriken und den Aufwandsabschätzungen.

5.1.4 Kritik und Messfehler

Auch wenn in diesem Projekt einige Störfaktoren, wie z.B. ein größeres Refactoring des Codes aufgefallen sind, kann das Analyseergebnis trotzdem als valide angenommen werden.

5.2 inGRID

Als zweiter Fall wird das interactive Generic Reporting Insight Dashboard (inGRID) System untersucht. Das inGRID-System ist eine von DXC entwickelte Anwendung, die eine fortschrittliche Monitoring- und Reporting-Lösung bietet. Sie wurde als Software as a Service Angebot mit einer zentralen Installation in der deutschen Cloud konzipiert. Eine Vielzahl von Agenten können verwendet werden, um Systeme zu überwachen. Die Agenten decken ein breites Spektrum an Überwachungsaktivitäten ab. Dazu gehören Virtual machine (VM)s, Middlewares, Anwendungen und andere Geräte (über SNMP). Die inGRID Anwendung wird von einem Offshore Team hauptsächlich für Kunden in Europa entwickelt. Das Entwicklungsteam ist in Ägypten lokalisiert und besteht aus neun Entwickler*innen. Davon haben vier Entwickler*innen mehr als drei Jahre Berufserfahrung und fünf Entwickler*innen weniger als drei Jahre Berufserfahrung. Die Entwickler*innen sind seit ungefähr zweieinhalb Jahren in dem Projekt involviert.

Das inGRID System besteht aus einer Vielzahl von individuellen Komponenten. Da die Analyse aller Komponenten für den Rahmen dieser Arbeit zu umfangreich wäre, wird hier lediglich die Backend Komponente des webbasierten Service Status Dashboards (SSD-Backend-Komponente) betrachtet. Während in der gesamten Anwendung eine Vielzahl von verschiedenen Programmiersprachen verwendet wird, besteht die SSD-Backend-Komponente zum Großteil (93%) aus Java Code.

InGRID wird agil nach Scrum in Sprints mit einer Dauer von jeweils drei Wochen realisiert. Innerhalb dieser drei Wochen werden Features von dem Backlog des Projektes umgesetzt. Der

reguläre Ablauf der Umsetzung von Features in dem inGRID Projekt konnte von einem der Entwickler erläutert werden. Der Ablauf beginnt mit der Anfrage eines neuen Features von einer, am Projekt beteiligten Person. Für dieses Feature wird nun eine User Story geschrieben und ein neuer Entwicklungszweig (englisch Branch) in der Quelltextverwaltung erstellt. In diesem neuen Entwicklungszweig wird nun neuer Quelltext hinzugefügt bzw. alter modifiziert. Wenn das Feature fertig entwickelt ist, wird es mit einem zentralen Entwicklungszweig verschmolzen (gemerged). Hier wird es möglichst innerhalb von 24 bis 48 Stunden vollständig getestet. Wenn alle Akzeptanzkriterien der User Story erfüllt sind, wird die User Story geschlossen.

5.2.1 Datenerhebung

Für die SSD-Backend Komponente der inGRID Anwendung soll nun wie auch in dem Fall der NDA Anwendung eine Zeitreihe der Komplexitätsmetriken und eine Zeitreihe der Aufwandsabschätzungen aufgestellt werden.

Zunächst wird hier die Datenerhebung für die Zeitreihe der Aufwandsabschätzungen beschrieben. Dies geschieht anhand User Storys. Von besonderem Interesse für die Analyse sind die Felder „Story Points“ und „Resolved Date“. Die Kombination beider Felder kann verwendet werden, um die Zeitreihe der Aufwandsabschätzungen aufzubauen. In dem Feld Story Points wird der Aufwand zur Realisierung der Story geschätzt. Mit dem, in 3.2.2 beschriebenen Planning-Poker-Verfahren wird der Umfang durch eine Kombination von Expertenmeinungen als eine Zahl in der Fibonacci Sequenz beschrieben. Abweichend von der in 3 beschriebenen Vorgehensweise werden in diesem Projekt jedoch auch die Erfahrung der Entwickler, sowie der abgeschätzte Aufwand in die Story Point Schätzungen mitaufgenommen. Das Feld „Resolved Date“ ist für die zeitliche Verortung der Aufwandsabschätzung der User Story relevant. Im Interview mit einem der Entwickler konnte ermittelt werden, dass dieses Feld den Zeitpunkt der Fertigstellung einer Story beinhaltet[190]. Es kann also mit diesem Feld gesagt werden, wann der, in der User Story spezifizierte Aufwand in die Anwendung geflossen ist. In dem Experteninterview[191] mit einem der Entwickler konnte an dieser Stelle eine Verbindung zwischen den Daten aus der Projektmanagementsoftware und den Messdaten aus dem Code geschlussfolgert werden. Wenn eine Story geschlossen wird, wird auch der Code dieser Story in den „develop“ Branch gemerged. Also lässt sich sagen, dass der Zeitpunkt in dem „Resolved Date“ der Story ungefähr dem Zeitpunkt der Änderungen in dem Quelltext der Anwendung entsprechen muss. Laut dem Entwickler betrügen die Abweichungen zwischen diesen beiden Zeitpunkten in mehr als 95% der Fälle weniger als 48 Stunden. In Relation zu der Gesamtdauer des Projektes kann diese Abweichung bei einem Signifikanzniveau von 5% als nicht signifikant angesehen werden. Die Verbindung beider Ereignisse über dieses Merkmal ist also valide.

Für die Analyse der User Storys anhand der zuvor beschriebenen Merkmale müssen die User Storys eine Reihe von Kriterien erfüllen. Anhand dieser Kriterien wird eine Vorauswahl der User

[190] Vgl. *Interview Mit Einem Entwickler Der inGRID Anwendung* 2022
[191] Vgl. *Interview Mit Einem Entwickler Der inGRID Anwendung* 2022

Storys getroffen. Dabei werden alle User Storys der Komponente „SSD Backend“ ausgewählt, die einen Status von „Done“ in dem Feld „Resolution“ haben, und denen eine Anzahl an Story Points zugewiesen ist. Jira bietet eine Möglichkeit zur genauen Abfrage von Storys mithilfe der proprietären Abfragesprache Jira Query Language (JQL). Die Syntax von JQL ist stark an die Syntax der Abfragesprache Structured Query Language (SQL) angelehnt[192]. Die JQL Abfrage für die User Storys findet sich im folgenden Code-Fragment.

```
1 project = HCC AND component = "SSD Backend" AND issuetype = Story AND
      resolution = Done AND "Story Points" is not EMPTY ORDER BY
    resolved DESC, priority DESC, updated DESC
```

Mit dieser Abfrage konnten in dem inGRID Projekt 62 User Storys ausgewählt werden. Diese wurden als CSV-Datei exportiert und in die Analysesoftware geladen.

Für die zweite Zeitreihe der Quelltextkomplexitätsmessungen wird ein Quelltextrepository benötigt. Mit der Unterstützung des Entwicklers des Projektes konnte das passende Repository ausgewählt werden. In dem Repository konnten in dem entsprechenden Entwicklungszweig „develop“ 2554 Commits identifiziert werden. Zu jedem dieser Commits kann der Stand des Quelltextes rekonstruiert werden. Anhand dieser Rekonstruktion kann, wie in 4.7.2 beschrieben eine Zeitreihe der Komplexitätsmessungen erstellt werden.

5.2.2 Auswertung

Im Sinne der Fallstudienforschung werden in der Auswertung alle Daten in Relation zueinander gebracht, um so zu logischen Schlüssen zu gelangen. Wie in 4.5 beschrieben, werden die Aufwandsabschätzungen zunächst in Relation zu den Komplexitätsmetriken gebracht. Die so erlangten Daten werden, wie in Kapitel 4.5 beschrieben und verarbeitet. Der auf diese Weise erlangte Korrelationsgraph ist in Abbildung TODO zu sehen und wird nun beschrieben.

TODO

In dem Graph sind die Story Points, logischen Codezeilen, die zyklomatische Komplexität und die Einrückungskomplexität genau wie in dem NDA Projekt (siehe 5.1.2) abgezeichnet. Die Differenz zwischen dem Durchschnitt aller Codemetriken und den Storypoint Abschätzungen findet sich, wie auch in dem ersten Projekt in einem separaten Diagramm unten in der Grafik.

Zunächst verlaufen die Storypoint-Abschätzungen nahe bei den Codekomplexitätsmetriken. Bis Ende Juni 2020 sind Abweichungen von 20 bis 30 Prozent ersichtlich. Anfang Juli 2020 ist in allen Codekomplexitätsmetriken ein plötzlicher Abfall der Werte zu beobachten. Dieser Abfall ist in den Aufwandsabschätzungen nicht vermerkt. In Zusammenarbeit mit dem Entwickler konnte eine Migration der Entwicklungsumgebung als möglicher Grund hierfür identifiziert werden[193]. Bis Januar 2021 fallen in den Codemetriken keine Veränderungen auf. Gleichzeitig steigen die

[192] Vgl. Ltd 2022
[193] Analyse Tabelle Zeile 38, Spalte K TODO

Komplexitätsabschätzungen in diesem Zeitraum aber kontinuierlich weiter. Eine Erklärung für diese Abweichung konnte nicht gefunden werden. Für die plötzlichen Anstiege und Abfälle der Codekomplexitätsmetriken von Januar 2021 bis März 2021 konnte ein Handover zwischen zwei Entwicklern als Erklärung gefunden werden. Von März bis August 2021 bleiben sowohl die Komplexitätsmessungen, als auch die Aufwandsabschätzungen größtenteils stabil. Diese Messdaten sprechen also für eine Korrelation. Ab August bis zum Ende der Messdaten im November 2021 lassen sich Anstiege in den Codekomplexitätsmetriken beobachten. Hier wurden bereits Funktionen in der Anwendung entwickelt, die aber noch nicht als geschlossene Storys in der Projektmanagementsoftware vermerkt sind.

Insgesamt verliefen in diesem Projekt die Aufwandsabschätzungen nur in Teilen ähnlich wie die Codekomplexitätsmetriken. Für die Abweichungen konnten nicht an allen Stellen Erklärungen gefunden werden. Es ist also zu erwarten, dass die berechneten Korrelationskoeffizienten nur eine geringe Korrelation anzeigen.

TODO

Diese Erwartung ließ sich durch eine Berechnung der Korrelationskoeffizienten bestätigen. Für die Komplexitätsmaßen der logischen Codezeilen und der Einrückungskomplexität konnten über die Koeffizienten hinweg nur Werte zwischen 0,64 und 0,43 erreicht werden. Diese Werte schließen zwar keine Korrelation aus, sprechen aber deutlich gegen eine Existenz dieser. Für die Komplexitätsmaßen der zyklomatischen Komplexität und des Halstead Aufwandes konnten deutlich höhere Werte zwischen 0,69 und 0,92 erreicht werden. Bei diesen Komplexitätsmaßen ist in diesem Projekt also eine Korrelation mit den Storypoint-Aufwandsabschätzungen durchaus anzunehmen. Als Erklärung für die Abweichung zwischen den Komplexitätsmaßen kann ihre Berechnungsmethodik vermutet werden. Sowohl die Maßzahl der logischen Codezeilen als auch die der Einrückungskomplexität orientieren sich, wie in 2.3 erklärt, zumindest in Teilen an den Zeilen im Sourcecode. Bei den anderen beiden Maßen spielt die Aufteilung des Codes auf Codezeilen keine Rolle. Hier wird lediglich der Inhalt des Codes betrachtet. Also könnte die Hypothese aufgestellt werden, dass eine Aufteilung oder nicht-Aufteilung des Quelltextes auf Codezeilen Einfluss auf die Maßzahl der logischen Codezeilen und der Einrückungskomplexität genommen haben könnte.

5.2.3 Fazit

Insgesamt konnte in diesem Projekt nur eine eingeschränkte Korrelation zwischen den Codekomplexitätsmetriken und den Aufwandsabschätzungen nachgewiesen werden. Für die Metriken der zyklomatischen Komplexität und die von Halstead konnten starke Korrelationen nachgewiesen werden. Hier spricht das Projekt also für eine generelle Korrelation. Bei den anderen beiden Metriken konnten aber nur schwache Korrelationen festgestellt werden, was wiederum gegen eine generelle Korrelation spricht. Insgesamt schwächt dieses Projekt also die Hypothese, dass Codekomplexitätsmetriken und Aufwandsabschätzungen zusammenhängen.

5.2.4 Kritik

In diesem Projekt wurden größere Abweichungen zwischen den Story Point Abschätzungen und den Codemetriken festgestellt. Die Existenz dieser Abweichungen deutet zunächst daraufhin, dass ein Fehler in der Messung existieren könnte. Des Weiteren konnten nicht für alle Abweichungen Erklärungen gefunden werden. Das könnte bedeuten, dass das Interview mit dem Entwickler nicht ausreichend umfangreich durchgeführt wurde. Insgesamt sollte die Validität der Ergebnisse dieses Falls stark hinterfragt werden.

5.3 Alstonii

Der zweite Fall dieser Fallstudie befasst sich mit dem Alstonii Projekt. Mit dem Alstonii Projekt soll ein mobiles Frontend für Vertriebsmitarbeiter eines großen Telekommunikationsanbieters geschaffen werden. Es soll diesen Mitarbeitern bei dem Verkauf von Breitband Internetanbindungen an Privat- und Firmenkunden unterstützen. Als digitales System soll das Alstonii Projekt einen analogen Prozess auf Basis von Papierformularen ersetzen. Ziel ist die Unterstützung der gleichen Funktionalität wie der analoge Prozess.

Das Projekt wird agil nach Scrum entwickelt und in mehreren Programmiersprachen umgesetzt. Dabei kommen zu 82% Vue, zu 10% TypeScript und zusätzlich (<5%) noch HTML, JavaScript, Python, CSS, SCSS, Shell und Batch zum Einsatz. Realisiert wird die Anwendung hauptsächlich von einem Nearshore Team in Bulgarien. Unterstützt wird dieses Team von einem Offshore Team in Indien und einigen Onsite Mitarbeitern. Insgesamt arbeiten so zwischen sechs und neun Mitarbeiter an dem Projekt. Hauptsächlich kommen erfahrene Mitarbeiter zum Einsatz, wobei die Erfahrung der Mitarbeiter in dem Projekt trotz der langen Laufzeit als gering einzustufen ist. Das lässt sich auf eine hohe Mitarbeiterfluktuation zurückführen. Das aktuelle Entwicklungsteam ist seit sechs Monaten in dem Projekt.

5.3.1 Datenerhebung

Auch in diesem Projekt konnten die Aufwandsabschätzungen aus der Projektmanagementsoftware Jira exportiert werden. Nach den in Kapitel 4.4 definierten Kriterien konnten 215 Datensätze identifiziert werden. Diese wurden als CSV-Datei exportiert und in die Analysesoftware eingelesen.

Als Quelle für die Komplexitätsmaßzahlen wurde das GitHub Repository der Software identifiziert. Es konnte ein Zugang beantragt werden und die Daten konnten über eine Secure Shell (SSH) Verbindung in eine lokale Kopie geladen werden. Diese Kopie wurde dann von der Analysesoftware gelesen.

5.3.2 Auswertung

In der Auswertung werden alle Daten nun miteinander verbunden, um so einen Schluss auf die Hypothese zu erlangen. Die Relation der Aufwandsabschätzungen mit den Komplexitätsmetriken ist in Abbildung TODO erkenntlich.

TODO Grafik

Das Projekt wird über einen Zeitraum von zwei Jahren von März 2020 bis Mai 2022 betrachtet. Eine dicke schwarze Linie zeigt die Aufwandsabschätzungen. Die restlichen, dünneren Linien zeigen die Komplexitätsberechnungen für die zyklomatische Komplexität, die Einrückungskomplexität, den Halstead Aufwand und die logischen Codezeilen. Rein visuell lässt sich erkennen, dass die Linien der Komplexitätsmetriken ähnlich zu den Aufwandsabschätzungen verlaufen. Es werden jedoch auch größere Abweichungen deutlich. Einige dieser Abweichungen konnten in einem Abschlussgespräch zu dem Projekt erklärt werden.

Zunächst konnte für die größeren Sprünge in der Einrückungskomplexität (Nummer 1) die Verwendung eines Linters als Erklärung herangezogen werden. Die Einrückungskomplexität wird über Einrückungen im Code berechnet. Diese werden durch einen Linter teils automatisiert in einem großen Umfang geändert. Der Linter passt hier die Einrückungen auf ein festes Niveau an, wodurch die Summe der Einrückungen (siehe Kapitel 2.3.5) in einem großen Umfang geändert wird. Bei den Sprüngen in der Einrückungskomplexität fällt auch auf, dass das Komplexitätsmaß nach dem Sprung wieder auf das gleiche Niveau wie vor dem Sprung zurückkehrt. Das kann dadurch erklärt werden, dass die Änderungen in der Einrückung durch den Linter nur temporär geschehen. Wird wieder zu dem ursprünglichen Einrückungsformat zurückgewechselt, ist die Summe der Einrückungen wieder auf dem alten Niveau.

Weiter ist von März 2021 bis Oktober 2021 ein Plateau in der zyklomatischen Komplexität zu beobachten (Nummer 3). Dieses Plateau lässt sich durch eine Fluktuation in der Besetzung des Projektteams erklären. So wurde hier der Stil der Entwicklung grundsätzlich geändert. Nach der Einarbeitungsphase des neuen Teams kam es zu einem Sprungartigen Anstieg in den Komplexitätsmetriken.

Als letzter auffälliger Punkt wurde ein plötzliches Abfallen mit einem anschließenden Abflachen der Komplexitätsmetriken zu Ende des Projektes ab März 2022 identifiziert. Als potenzielle Erklärung hierfür wurde eine Verschiebung des Entwicklungszieles vorgeschlagen. So wurde zum Ende des Projektes ein verstärkter Fokus auf die Konsolidierung der Codebasis und auf das Verbessern der Leistung der Applikation gelegt. Bei einer Überarbeitung der Applikation ist zu erwarten, dass der Umfang der Anwendung gleichbleibt. Diese Erwartung konnte mit den vorliegenden Messdaten bestätigt werden.

Zusammenfassend entsprechen die hier gesammelten Daten zwar in grobem Maße der Hypothese, es sind jedoch einige signifikante Abweichungen zwischen den Maßen und den Aufwandsabschätzungen erkenntlich. Zu einem Teil konnten sie durch technische Störfaktoren erklärt werden. Zu

einem anderen Teil sind sie aber durch Eigenheiten in dem Verlauf des Projektes zu erklären. Nach dieser Durchsicht der Messdaten ist also eine weniger starke Korrelation als z.B. in dem NDA Projekt zu erwarten. Die Korrelation der Komplexitätsmetriken mit den Aufwandsabschätzungen wird nun berechnet.

TODO

Für alle Untersuchungsergebnisse wurde ein P-Wert von $< .00001$ ermittelt. Bei einem, für Sozial- und Wirtschaftswissenschaften üblichen Signifikanzeniveau von .05 lässt sich also sagen, dass die Ergebnisse signifikant sind.

Wie zu erwarten war, wurden in diesem Fall geringfügig niedrigere Korrelationskoeffizienten ermittelt als z.B. in dem NDA Projekt ermittelt. Im Durchschnitt sind die Werte aber nur um etwa 5% geringer als in dem NDA Projekt.

5.3.3 Fazit

Auch in dem Alstonii Projekt konnten erfolgreich Daten entsprechend den zuvor gestellten Anforderungen gesammelt werden. Diese Daten konnten ebenfalls erfolgreich in einen Zusammenhang mit der These gestellt werden. Trotz stärkerer Störfaktoren wurde hier über Korrelationkoeffizienten und Komplexitätsmetriken hinweg eine starke Korrelation festgestellt. Also spricht dieser Fall für eine Bestätigung der hypothetisierten Korrelation zwischen Softwarekomplexitätsmetriken und Aufwandsabschätzungen.

5.3.4 Kritik

In diesem Projekt ist die Bedeutung der Störfaktoren deutlich größer als in dem NDA Projekt. Die Abweichungen zwischen den Komplexitätsmaßen und den Aufwandsabschätzungen ließen sich nicht ausnahmslos erklären.

5.4 Lacustris

Ein weiteres Projekt ist das Lacustris Projekt[194]. Hierbei handelt es sich um einen Microservice, welcher ein Teil eines größeren Projektes darstellt. Der Microservice ist dafür verantwortlich, Daten aus einer Amazon Web Services (AWS) Instanz abzurufen und diese für andere Microservices zur Verfügung zu stellen[195]. Entwickelt wird dieses Projekt ebenfalls in einer agilen Arbeitsweise, sowohl von onshore, als auch von offshore Mitarbeitern. Primär sind die Entwickler in Ägypten lokalisiert[196]. Die Teams bestehen dabei aus einem Mix aus junior und senior Mitarbeitern. Alle

[194] Lacustris Zeile 3
[195] Lacustris Zeile 4
[196] Lacustris Zeile 7 und 8

Mitarbeiter sind erst seit wenigen Monaten an dem Projekt beteiligt. Die Programmiersprache des Projektes ist ausschließlich Java.

5.4.1 Datenerhebung

Zum Berechnen der Korrelationen sollen auch in diesem Projekt die Aufwandsabschätzungen und die Codekomplexitätsmetriken erhoben werden.

Die Aufwandsabschätzungen lassen sich in diesem Projekt ähnlich wie in den anderen Projekten der Projektmanagementsoftware Jira entnehmen. Dabei liegen die Story Point Abschätzungen sowie ein Zeitstempel in einem Feld names „Resolved“ vor. In diesem Projekt konnten insgesamt 58 relevante User Storys gefunden werden

Auch zu dem Git Repository des Projektes konnte ein Zugang erhalten werden. Hier konnte die Entwicklungsgeschichte des Projektes an 965 Zeitpunkten rekonstruiert werden.

5.4.2 Auswertung

Für die Auswertung werden nun auch in diesem Projekt die Zeitreihe der Aufwandsabschätzungen mit der der Codekomplexitätsmetriken in Verbindung gebracht. Der daraus resultierende Korrelationsgraph ist in Abbildung TODO erkenntlich.

TODO Grafik

Die Verläufe der verschiedenen Messreihen sollen nun beschrieben und Anhand der Hinweise aus dem Abschlussinterivew des Projektes erläutert werden. Die verschiedenen Ereignisse werden dabei in die Kategorien People, Process und Technologies gegliedert.

In der Kategorie Process fallen vier Ereignisse auf. Alle diese Ereignisse beziehen sich auf die Projektphasen des Projektes. Vom Anfang des Projektes am 22. Juli 2021 bis Mitte August 2021 fällt eine vergleichsweise geringe Produktivität auf. Hier wurde als Begründung herbeigeführt, dass sich die Dynamik innerhalb des Teams erst bilden musste, und das Team somit noch nicht so produktiv sein konnte (#1). Von Mitte August 2021 bis Mitte November 2021 ist die Produktivität des Teams deutlich höher und das Team befindet sich in einer Phase der kontinuierlichen Entwicklung (#2 und #4). Ab November 2021 geht das Projekt in eine Phase der Stabilisierung über und die Codebasis wird eher konsolidiert als erweitert. Hier steigen die Metriken also weniger stark (#5).

In der Kategorie People fallen drei Ereignisse auf. Zunächst war in der letzten Woche des Novembers das komplette ägyptische Team im Urlaub (#7). Dem folgte im Januar 2022 das Englische Team (#8). Als drittes Ereignis wurde Mitte Februar 2022 eine Entwicklerin von dem Projekt entlassen (#6). Diese drei Ereignisse wirkten sich kaum auf den Projektverlauf aus, da sich das Projekt hier schon in einer Phase der Stabilisierung befand.

Neben diesen Projektereignissen fallen vom 4. bis zum 13. September 2021 starke Schwankungen in allen Metriken auf. Nach diesen Schwankungen verblieben die Metriken für den Rest der Projektdauer auf einem, um 20% höheren Stand als vorher. Dieser starke Anstieg ließ sich auf das Hinzufügen von vorgeneriertem Code zurückführen. So wurden durch ein externes Programm Datenmodelle generiert, welche an dieser Stelle dem Code hinzu gefügt wurden[197].

In einem nächsten Schritt werden die Korrelationsmetriken berechnet:

TODO Grafik

Die Spearman Rangkorrelationskoeffizienten sind über alle Metriken hinweg sehr stark mit Werten über 0,99. Die Rangkorrelationskoeffizienten nach Kendall liegen mit Werten um 0,95 nur leicht unter den Werten von Spearman. Auffallend ist bei beiden Koeffizienten, dass die die Werte der Korrelationskoeffizienten über die Metriken hinweg konstant bleiben. Die Pearson-Produkt-Moment-Korrelationskoeffizienten liegen im Schnitt deutlich unter den anderen Koeffizienten. Nur bei dem Aufwand nach Halstead konnte hier ein höherer Wert erzielt werden. Das könnte darauf zurückzuführen sein, dass der Störfaktor des vorgenerierten Codes in dem Halstead Aufwand nicht so stark widergespiegelt wird. Dass bei den anderen Koeffizienten der Halstead Aufwand nicht heraussticht deutet darauf hin, dass die Koeffizienten nach Kendall und Spearman von dem Störfaktor nicht so stark beeinflusst wurden.

5.4.3 Kritik

Klar zu kritisieren ist an der Analyse dieses Projektes, dass durch den vorgenerierten Code Mitte September 2021 eine starke Verfälschung der Messergebnisse stattgefunden hat. Die künstliche Erhöhung der Metriken könnte die Korrelation zwischen den Aufwandsabschätzungen und den Komplexitätsmetriken schwächen

5.4.4 Fazit

Insgesamt lassen sich in diesem Projekt starke Korrelationen zwischen den Aufwandsabschätzungen und den Komplexitätsmetriken nachweisen. Eine niedrigere Korrelation nach Pearson könnte auf den Störfaktor des vorgenerierten Codes zurückzuführen sein. Es lässt sich also abschließend sagen, dass dieser Fall für eine generelle Korrelation von Aufwandsabschätzungen und Codekomplexitätsmetriken spricht.

[197] LM Parties Zeile 19

5.5 Engelmannii

Das Engelmannii Projekt[198] wird von der DXC für einen deutschen Energieversorger entwickelt. Es soll den Anwendern als zentrale Verwaltungsschnittstelle für alle Dienste des Energieversorgers dienen[199]. Realisiert wird das Projekt in einer agilen Arbeitsweise nach Scrum sowohl von offshore, als auch onshore Teams. Das Offshore Team ist in Manila lokalisiert und das Onshore Team über Deutschland verteilt. Die Teams bestehen aus Entwicklern aller Erfahrungslevels, wobei alle seit mindestens zwei Jahren an dem Projekt arbeiten. Aufgebaut ist das Projekt aus einem Frontend und einem Backend. Das Frontend wird primär auf JavaScript aufgebaut mit Technologien wie Angular und Typescript. Das Backend wird mit .NET realisiert. Insgesamt ist die Codebasis des Projektes über drei Repositories verteilt. Ein Repository beinhaltet sämtlichen Frontend Code, eines den Code für die Application Programming Interface (API) Middleware in C# und ein letztes Repository umfasst eine sog. Progressive Web App.

5.5.1 Datenerhebung

Auch in dem Engelmannii-Projekt sollen einerseits die Aufwandsabschätzungen und andererseits die Quelltext Repositories als Primärquellen verwendet werden.

Die Aufwandsabschätzungen werden in dem Ernergieportal-Projekt in der Software LifeCycle Management Software Microsoft DevOps verwaltet. Das Format der Userstorys entspricht dabei zwar nicht in Gänze dem Format der anderen Projekte, die relevanten Teilinformation können aber auch hier problemlos abgelesen werden.

In einem Interview mit einem Stakeholder des Projektes[200] konnten die relevanten Teilbereiche der User Storys identifiziert werden. Die Aufwandsabschätzungen seien in einem Feld mit dem Namen „Front End Dev Story Points“ zu finden. Das Abschlussdatum jedes Features sei dem Feld „Closed Date“ zu entnehmen. Zusätzlich konnte eine Abfrage zur Identifizierung aller relevanten User Storys erstellt werden. Dabei schränkt die Abfrage die Auswahl der User Storys auf alle Elemente mit einem Wert für das Feld „Front End Dev Story Points“ und einem Status von „Closed“ ein. Von insgesamt 2329 User Storys wurden 588 als relevant ausgewählt.

Die so ausgewählten Userstorys können als CSV-Datei exportiert und in die Analysesoftware eingelesen werden.

Zur Ermittlung der Codekomplexitätsmetriken ist, wie in Kapitel 4.5 beschrieben eine Analyse des Quelltextes der Anwendung angestrebt. Wie auch in den anderen Projekten, soll ein Änderungsverlauf der Komplexitätsmetriken aus der Versionshistorie des Quelltextes erstellt werden. Diese Versionshistorie ist für alle Quelltext Repositories der Software im Format eines Git Repositories verfügbar. Wie eingangs beschrieben besteht das Projekt aus drei Repositories. In dieser

[198] Engelmannii Zeile 5 TODO
[199] Zeile 6
[200] Vgl. *Interview Mit Stakeholdern Des Englemanii Projektes* 2022

Analyse wird jedoch lediglich die Entwicklung am Frontend der Applikation betrachtet. Also ist auch nur das Repository des Frontends von Interesse. Mit der in 4.7. beschriebenen Software kann ein Zeitverlauf der Codekomplexitätsmetriken aus der Versionshistorie des Sourcecodes erstellt werden. Die so ermittelten Daten werden nun ausgewertet.

5.5.2 Auswertung

Zur Auswertung werden auch in diesem Projekt alle Daten in ein Verhältnis zueinander gesetzt. Das Verhältnis der Storypoint Aufwandsabschätzungen zu den Komplexitätsmaßzahlen findet sich in Abbildung TODO

TODO Grafik

Bei einer genaueren Betrachtung der Messwerte fällt auf, dass die zyklomatische Komplexität und die Einrückungskomplexität sehr nahe den Story Point Abschätzungen verlaufen. Die Anzahl der logischen Codezeilen und der Aufwand nach Halstead weisen jedoch von Mai 2019 bis August 2020 eine signifikante Abweichung von den Story Point Metriken von mehr als 300% auf. Für diese Abweichung muss ein Fehler in der Datenerhebung verantwortlich gemacht werden. Dabei wurden gewisse Berichtsdateien in die Analyse eingeschlossen, die aber nicht zu dem eigentlichen Quelltext der Anwendung dazugehören[201]. Im August 2020 fallen die Anzahl der logischen Codezeilen und der Halstead Aufwand stark ab. Nach diesem Abfall befinden sie sich wieder auf einem ähnlichen Niveau wie die anderen Metriken. Hier wurde in einem Interview als Erklärung gefunden, dass im August 2020 die zusätzlichen Berichtsdateien aus dem Quelltextrepository entfernt wurden. Des weiteren ist ab März 2021 ein Abflachen des Anstieges der Komplexität bei einem gleichzeitigen weiteren Anstieg der Aufwandsabschätzungen zu beobachten. Ab März 2021 wurde an einer Grundlegenden Umstrukturierung der Anwendung gearbeitet. Diese wurde jedoch in einem separaten Entwicklungszweig durchgeführt und konnte deswegen in der Analyse der Codemetriken nur begrenzt erfasst werden. Der Aufwand dieser Umstrukturierung ist aber gleichzeitig in den Aufwandsabschätzungen zu sehen. Das erklärt die Abweichung der beiden Werte in diesem Zeitraum. Für die frequenten kleineren Sprünge in der Einrückungskomplexität ist ein ähnlicher Messfehler wie in dem Alstonii Projekt verantwortlich.

TODO Grafik

Wie zu erwartet fallen die Korrelationskoeffizienten für die Anazhl der logischen Codezeilen äußerst gerin aus. Mit Werten unter 0,25 ist hier keine Korrelation nachweisbar. Das ist auf die Störung der Messung durch Berichtsdaten zurückzuführen. Hier wurde die Messung über ein Viertel der Projektdauer um 300% erhöht, was den geringen Korrelationskoeffizienten erklärt. Ähnlich wurde auch für den Aufwand nach Halstead ein sehr geringer Korrelationkoeffizient von 0,5 ermittelt. Für die anderen beiden Komplexitätsmetriken (Zyklomatische Komplexität und Einrückungskomplexität) konnten sehr starke Korrelationen nachgewiesen werden. Das unterstützt

[201] Vgl. *Interview Mit Stakeholdern Des Englemanii Projektes* 2022

wiederrum die Vermutung, dass die geringen Korrelationskoeffizienten der anderen Metriken mit dem Störfaktor der Berichtsdaten zu begründen sind.

5.5.3 Fazit

Zusammenfassend konnten in diesem Fall nur bei der zyklomatischen Komplexität und der Einrückungskomplexität eine ausreichend starke Korrelation der Codekomplexitätsmetriken mit den Aufwandsabschätzungen festgestellt werden. Aufgrund eines starken Störfaktors konnte bei den anderen beiden Metriken keine Korrelation festgestellt werden. Insgesamt sind zwar auch in diesem Projekt Korrelationen zwischen Codekomplexitätsmetriken und Aufwandsabschätzungen feststellbar jedoch fehlt es dem Fall für eine eindeutige Unterstützung der Hypothese an Aussagekraft.

5.5.4 Kritik

Klar zu kritisieren ist in diesem Fall, dass die Metriken des Halstead Aufwandes und der logischen Codezeilen durch einen Störfaktor sehr stark beeinträchtigt wurden.

5.6 GitLab

Als fünftes und letztes Projekt wurde die DevOps Software GitLab untersucht. Die GitLab Software ist ein Open-Source Tool zum Entwickeln, Absichern und Betreiben von Softwareanwendungen[202]. Besonders zu bemerken ist an dieser Stelle, dass die Firma eine besonders transparente Strategie zum Veröffentlichen von Informationen verfolgt[203]. Nach dieser Strategie werden alle firmeninternen Informationen zunächst publiziert, sofern für sie keine Ausnahmeregelung besteht. Eine Liste an Ausnahmen von dieser Richtlinie ist ebenfalls publiziert. So werden zum Beispiel Informationen zu der finanziellen Lage der Firma sowie Informationen zu Kunden nicht publiziert[204].

Entwickelt wird die GitLab Applikation von über 1500 Mitarbeitern, welche ohne physische Büros aus 66 Ländern verteilt arbeiten[205]. Die gesamte Anwendung umfasst über 650 Tausend Codezeilen und wird hauptsächlich in den Sprachen Ruby (66%) und JavaScript (20%)[206] geschrieben. Sämtlicher code der Anwendung wird in einem zentral Git Repository verwaltet.

[202] Vgl. *GitLab 14 Delivers Modern DevOps in One Platform* 2021
[203] Vgl. *GitLab Values* 2022
[204] Vgl. *GitLab Communication* 2022
[205] Vgl. *About GitLab | GitLab* 2022
[206] Vgl. *GitLab.Org / GitLab · GitLab* 2022

5.6.1 Datenerhebung

Aufgrund der Transparenten Kommunikation der GitLab gestaltet sich eine Datenerhebung in diesem Projekt vergleichsweise einfach. Wie auch in den anderen Projekten werden in diesem Projekt zwei Datenquellen benötigt. Zum einen sollen die Codekomplexitätsmetriken aus der Quelltextverwaltung der Software berechnet werden. Weiter sollen die Aufwandsabschätzungen aus der Projektmanagementsoftware des Projektes geladen werden. Beide Quellen sind inmFalle der GitLab Anwendung öffentlich zugänglich.

Das Quelltext Repository kann unter der Adresse `https://gitlab.com/gitlab-org/gitlab` abgerufen werden. Für die Analyse wurde von dieser Adresse eine lokale Kopie des Repositories geladen und in die Analysesoftware eingespeist. Über die Projektdauer von fünf Jahren konnten 281.414 Entwicklungsstände identifiziert werden.

Die Aufwandsabschätzungen werden in dem GitLab Projekt im Rahmen von Issues festgehalten. Issues entsprechen in ihrer Struktur in diesem Kontext den User Storys. Der Aufwand zum Realisieren eines Issues wird als das Gewicht des Issues angegeben. Es kann dabei Synonym zu den Story Point Abschätzungen gesehen werden[207]. Alle Issues des GitLab Projektes sind auf einer Website öffentlich verfügbar und lassen sich als CSV-Datei exportieren[208]. Auf diese Weise konnten 68.366 Datensätze erhoben werden. Die CSV-Datei mit diesen Datensätzen kann unverarbeitet in die Analysesoftware geladen werden.

Insgesamt konnten alle, für die Analyse benötigten Daten erfolgreich aus öffentlich verfügbaren Quellen erhoben werden und ohne eine Modifikation in die Analysesoftware geladen werden.

5.6.2 Auswertung

Die gesammelten Daten sollen nun ausgewertet werden. Der Umfang der Auswertung ist in diesem Projekt ca. 15-mal so groß wie in allen anderen Projekten. Aus diesem Grund muss die Auswertungsmethodik abgeändert werden. Die Berechnung der Entwicklungsstände der Software wird in einem Cluster aus vier Rechnungseinheiten in einem Rechenzentrum durchgeführt. Zusätzlich wurden lediglich die Module Lizard und MultiMetric des Code Parsers aktiviert. Auch mit diesen Änderungen dauerte die Analyse des Projektes mehrere Wochen. Aufgrund des begrenzten zeitlichen Umfangs dieser Arbeit konnte nur eine Analyse der ersten dreieinhalb Jahre des Projektes erfolgen. Das Ergebnis dieser Analyse ist in Abbildung .. gezeigt.

TODO Grafik

Zunächst ist visuell eine starke Korrelation aller Komplexitätsmetriken mit den Aufwandsabschätzungen erkenntlich. Gleichzeitig fallen jedoch auch kurzfristige Schwankungen in den Messdaten von ca. 10% auf. Diese Schwankungen sind über die Projektdauer hinweg in regelmäßigen

[207] Vgl. *Scaled Agile and GitLab* 2022
[208] Vgl. *Issues · GitLab.Org / GitLab · GitLab* 2022

Abständen zu beobachten. Bei einer genaueren Analyse wurde festgestellt, dass diese Schwankungen wahrscheinliche auf eine Asynchronität in der Berechnungsmethodik der vier Berechnungsknoten zurückzuführen sind. Dementsprechend handelt es sich hierbei also mit einer hohen Wahrscheinlichkeit um einen Messfehler. Trotz dieser Schwankungen ist ein hoher Korrelationsgrad zu erwarten. Dieser wird nun berechnet

TODO Grafik

Entsprechend der Erwartungen wurden in diesem Projekt starke, konsistente Korrelationen zwischen allen Metriken und den Aufwandsabschätzungen festgestellt. Würde man den Messfehler beheben, ist anzunehmen, dass die Korrelationen noch stärker ausfallen.

5.6.3 Kritik

An der Analyse dieses Projektes fallen mehrere Kritikpunkte auf. Zum einen unterliegt die Analyse potenziell einem Messfehler, welcher zu Schwankungen in den Messdaten führt. Eine Behebung dieses Messfehlers war aufgrund des begrenzten zeitlichen Rahmens dieser Arbeit nicht möglich. Zusätzlich konnte in dem Projekt kein Abschlussinterview zu Validierung der Daten durchgeführt werden. Die Validität des Datenerhebungsverfahren konnte nur aus den öffentlich verfügbaren Informationen der Firma begründet werden.

5.6.4 Fazit

Zusammenfassend wurde in diesem Projekt eine starke Korrelation der Aufwandsabschätzungen mit den Codekomplexitätsmetriken festgestellt. Diese Korrelation könnte potenziell noch stärker ausfallen, wenn die potenziellen Messfehler in den Daten behoben werden könnten. Insgesamt lässt sich also sagen, dass dieser Fall die Hypothese einer Korrelation zwischen Aufwandsabschätzungen und Codekomplexitätsmetriken stützt.

6 Zusammenfassende Analyse der Untersuchungsergebnisse

Im vorherigen Kapitel wurde die Korrelation zwischen Aufwandsabschätzungen und Codekomplexitätsmetriken in sechs Fällen untersucht. Das Ergebnis dieser Untersuchung ist ein vier-Dimensionales Datenset. Die vier Achsen dieses Datensets sind das Projekt, die Codekomplexitätsmetrik, der Korrelationskoeffizient und der Wert dieser Korrelation. Die Darstellung vierdimensionaler Daten ist zwar möglich[209], im Rahmen dieser Arbeit aber wenig praktikabel. Also ist die Reduktion des Datensets auf drei Dimensionen angestrebt. Es muss also eine Dimension entfernt werden. Da das Ziel dieser Arbeit eine Aussage über eine generelle Korrelation ist, wird die Dimension der Korrelationskoeffizienten vernachlässigt. Auch wurde diese Dimension in der Betrachtung der individuellen Projekte bereits analysiert. Zur Reduktion des Datensets auf drei Dimensionen wird das arithmetische Mittel der drei Korrelationskoeffizienten als Wert übernommen. Das reduzierte Ergebnis aller Projekte ist in Abbildung TODO gezeigt.

TODO Grafik

In Abbildung TODO sind die durchschnittlichen Korrelationskoeffizienten für die einzelnen Komplexitätsmetriken in den Projekten dargestellt. Zunächst wird versucht die Ergebnisse in den individuellen Projekten zu begründen. Dann soll ein genereller Schluss gezogen werden.

Der Fall des DIL NDA Projektes weist in allen vier Komplexitätsmetriken starke Korrelationen auf. In der Analyse wurden kaum Störfaktoren identifiziert. Es ist also davon auszugehen, dass das Ergebnis hinreichend genau ist. Dieser Fall spricht für eine Korrelation aller vier Komplexitätsmetriken mit den Aufwandsabschätzungen.

Im Falle des inGRID-Projektes konnten für die Zyklomatische Komplexität und den Aufwand nach Halstead starke Korrelationen nachgewiesen werden. Für das Fehlen der Korrelationen bei den Logischen Codezeilen und der Einrückungskomplexität konnte keine Erklärung gefunden werden. Also spricht dieser Fall gegen Korrelation zwischen diesen beiden Metriken und den Aufwandsabschätzungen.

Das Alstonii Projekt spricht trotz einiger Störfaktoren für eine Korrelation zwischen allen Metriken und den Aufwandsabschätzungen.

Auch in dem Lacustris Projekt konnten für alle Metriken sehr starke Korrelationen festgestellt werden. Die Pearson-Produkt-Moment-Korrelation fiel für die Anzahl logischer Codezeilen, die zyklomatische Komplexität und die Einrückungskomplexität zwar nur schwach aus, jedoch ist diese schwache Korrelation wahrscheinlich auf einen Störfaktor zurückzuführen.

[209] Sarkar 2021

Bei dem Engelmannii Projekt korrelieren nur die zyklomatische Komplexität und die Einrückungskomplexität mit den Aufwandsabschätzungen. Die beiden anderen Metriken wurden von einem Messfehler gestört.

Zuletzt konnten in dem Fall des GitLab Projektes ebenfalls sehr starke Korrelationen aller Komplexitätsmetriken gemessen werden.

In Tabelle TODO findet sich eine Übersicht aller Komplexitäts-Korrelationen aller Projekte. Ein X in dem entsprechenden Feld bedeutet, dass hier durchschnittlich ein starker Korrelationskoeffizient (> 0.8) festgestellt wurde.

TODO Grafik

Insgesamt konnte für die zyklomatische Komplexität in allen Fällen eine starke Korrelation nachgewiesen werden. Für den Aufwand nach Halstead und die Einrückungskomplexität konnte in 5 von 6, also 80% der Fällen eine Korrelation nachgewiesen werden. Bei der Anzahl logischer Codezeilen wurde nur in vier von sechs Fällen, also zu 70% eine Korrelation nachgewiesen.

Die Ausgangsfragestellung dieser Arbeit ist, ob eine Korrelation zwischen Storypoint Aufwandsabschätzungen und Softwarekomplexitätsmetriken existiert. Diese Frage kann mit einem eingeschränkten „Ja“ beantwortet werden: Bei der Komplexitätsmetrik der zyklomatischen Komplexität sprechen alle Fälle für eine Korrelation. Zwar kann aufgrund der geringen Anzahl an Fällen in dieser Studie keine generelle und allgemeingültige Aussage zu der Korrelation der zyklomatischen Komplexität mit den Aufwandsabschätzungen getroffen werden, jedoch ist eine solche Korrelation sehr wahrscheinlich. Bei dem Halstead Aufwand und der Einrückungskomplexität spricht jeweils ein Projekt gegen eine Starke Korrelation. Trotzdem ist eine generelle Korrelation durchaus wahrscheinlich. Zuletzt sprechen bei der Anzahl an logischen Codezeilen nur 70% der Fälle für eine starke Korrelation der Metrik mit den Aufwandsabschätzungen, jedoch ist auch hier eine leichte Korrelation sehr wahrscheinlich. Insgesamt kann mit den Ergebnissen dieser Arbeit gesagt werden, dass eine Korrelation zwischen Story Point Aufwandsabschätzungen und Codekomplexitätsmetriken nicht ausgeschlossen und in den meisten Fällen sehr wahrscheinlich ist. Die anfängliche Zielsetzung einer Näherungsangabe zu dieser Frage ist also hiermit erfüllt.

7 Conclusion

Die vorliegende Bachelorarbeit befasst sich mit der Fragestellung, ob eine Korrelation zwischen Story Point Aufwandsabschätzungen und Softwarekomplexitätsmetriken besteht. Das Ziel dieser Arbeit ist eine Näherungsangabe als Antwort zu dieser Frage zu formulieren. Als Forschungsmethodik wird eine explorative Fallstudien über sechs Softwareprojekte durchgeführt.

Zur Beantwortung der Fragestellung wird zunächst die Theorie der Softwarekomplexitätsmetriken erläutert. Die Softwarekomplexitätsmetriken werden als Teil der statischen Code-Analyse in der Softwarequalitätssicherung in den Software Lebenszyklus eingeordnet. Dann werden sie als Verfahren zur Quantifizierung der Vielschichtigkeit der Struktur eines Computerpro-gramms definiert. Es wird auf die umfangreiche Kritik an Softwarekomplexitätsmetriken eingegangen. So seien diese Metriken schwach in ihrer Aussagekraft und könnten die subjektiv wahrgenommene Komplexität einer Software nicht quantifizieren. Im nächsten Unterkapitel der theoretischen Abhandlung der Softwarekomplexitätsmetriken wird eine Auswahl von Metriken getätigt. Die Metriken Anzahl von logischen Codezeilen, Zyklomatische Komplexität und Halstead Aufwand werden aufgrund ihrer hohen Anzahl an Zitierungen in Fachliteratur ausgewählt. Die Metrik Einrückungskomplexität wird aufgrund ihrer einzigartigen Berechnungsmethode betrachtet. Für alle vier Metriken wird dann ihr Ursprung und ihre Berechnungsmethodik erläutert. Außerdem wird an dieser Stelle noch einmal auf Kritik an den individuellen Metriken eingegangen. Als zweite Vergleichsgröße werden auch die Story Point Aufwandsabschätzungen erklärt. So wird im Kontext der agilen Softwareentwicklung nach Scrum der Aufwand zur Realisierung einer User Story, bzw. eines Features vor der Entwicklung abgeschätzt.

Nach der theoretischen Abhandlung der Softwarekomplexitätsmetriken und der Aufwandsabschätzungen folgt nun eine Beschreibung des Forschungsaufbaus. Es wird eine explorative Fallstudie an sechs Softwareprojekten durchgeführt. In jedem Projekt werden zwei Datenquellen untersucht. Als erste Quelle werden die Aufwandsabschätzungen aus der Projektmanagementsoftware extrahiert. Die zweite Quelle sind die Quelltextverwaltungen der Projekte. Aus der Versionshistorie des Quelltextes können die Softwarekomplexitätsangaben für die einzelnen Entwicklungsstände der Software berechnet werden. Aus beiden Quellen wird jeweils eine Zeitreihe der Messwerte aufgestellt. Beide Zeitreihen werden verbunden und es wird ein Korrelationskoeffizient zwischen ihnen berechnet. Dieser Korrelationskoeffizient bezeichnet die Korrelation zwischen den Softwarekomplexitätsmetriken und den Aufwandsabschätzungen. Er wird als Ergebnis der Fälle betrachtet. Die Berechnung des Korrelationskoeffizienten aus den Quellen erfolgt mit einer Analysesoftware. Die Implementierung dieser Analysesoftware wird ebenfalls beschrieben.

In einem nächsten Schritt findet die Untersuchung der sechs Softwareprojekte als Fälle der Fallstudie statt. Im ersten, dritten, fünften und sechsten Fall kann eine starke Korrelation aller Komplexitätsmetriken mit den Aufwandsabschätzungen gezeigt werden. Im zweiten Fall und dritten Fall hingegen sind nur starke Korrelationen zweier Metriken nachweisbar.

Insgesamt konnte durch die Fallstudie das Forschungsziel einer Näherungsangabe zu der Korrelation von Softwarekomplexitätsmetriken und Aufwandsabschätzungen erreicht werden. In allen Fällen korreliert die zyklomatische Komplexität stark mit den Aufwandsabschätzungen. Also ist eine allgemeine Korrelation wahrscheinlich. Für den Aufwand nach Halstead und die Einrückungskomplexität konnte in 5 von 6, also 80% der Fällen eine Korrelation nachgewiesen werden. Bei der Anzahl logischer Codezeilen wurde nur in vier von sechs Fällen, also zu 70% eine Korrelation nachgewiesen.

Insgesamt lautet das Ergebnis, dass eine starke Korrelation ($> 0{,}8$) der Softwarekomplexitätsmetriken mit den Aufwandsabschätzungen in jedem Fall wahrscheinlich, in keinem Fall aber garantiert ist.

7.1 Kritische Evaluierung

Die Ergebnisse dieser Arbeit werden nun kritisch evaluiert. Insbesondere wird dabei auf die Gütekriterien zur Bewertung von Fallstudien nach Gothlich 2003 genommen[210].

An der *Objektivität*[211] der Fallstudie lässt sich kritisieren, dass die Quelldaten der Analyse nicht als Teil der Arbeit veröffentlicht wurden. Diese beinhalten teils vetrauliche Daten und können deswegen nicht veröffentlicht werden. Die Analysemethoden kann jedoch anhand der öffentlichen Datenquellen des GitLab falls überprüft werden.

Die *validität der Analysekonstrukte* wurde zwar nicht in einem Gutachtenstil bewiesen, in einigen der Projekte jedoch durch Projektbeteiligte validiert.

Auch an der *internen Validität* der Arbeit gibt es Kritikpunkte. So wurde in der Interpretation der Daten keinerlei alternative Interpretationsmöglichkeiten berücksichtigt.

Zu der *externen Validität* der Fälle lässt sich sagen, dass zwar eine Replikationslogik angewendet wurde, das Analysevorgehen jedoch nicht durch Feedbackschleifen mit den Ergebnissen der Fälle verfeinert wurde.

Für die *Reliabilität* spricht zunächst das sämtliche Analysen vollständig automatisiert durchgeführt wurden, was Rechenfehler größtenteils ausschließt. Jedoch werden dadurch eventuell auch eigenheiten in den Messdaten ignoriert. So sind in mehreren der Fälle Messfehler aufgefallen, die auf einer Fehlinterpretation der Daten durch die Analysesoftware beruhen.

Zuletzt ist auch die *Utilitarität* der Studie gegeben. Abgesehen von einem regen Interesse innerhalb der Firma steht dem Aufwand zur Realisierung der Studie auch der generelle Nutzen der Validierung der Komplexitätsmetriken gegenüber.

Zusätzlich zu den Gütekriterien von Gothlich werden auch weitere Kritikpunkte

[210] Vgl. Göthlich 2003, S. 13
[211] Vgl. Göthlich 2003, S. 13

7.2 Ausblick

Im Folgenden soll aufbauend auf den Ergebnissen dieser Arbeit ein Ausblick auf mögliche weitere Entwicklungen gegeben werden.

7.2.1 Zusätzliche Validierung der Ergebnisse

Zunächst können die hier erlangten Ergebnisse weiter validiert werden. Dazu sollten einerseits die Messfehler in der Untersuchung der Projekte behoben werden. Hierzu könnte die Analysesoftware verbessert werden. Auch kann die Validierung der Ergebnisse durch mehr und umfangreichere Interviews mit Projektbeteiligten verbessert werden. Zuletzt würde eine Ausweitung der Fallstudie auf mehr Projekte die Validität der Gesamtaussage weiter stützen.

7.2.2 Umgebungsparameteranalyse

Neben der Weiterentwicklung der Korrelationsanalyse ist im Rahmen dieser Arbeit noch ein weiterer interessanter Zusammenhang aufgefallen. An den Stellen, an denen die Codekomplexitätsmetriken besonders stark von den Aufwandsabschätzungen abwichen ließen sich in vielen Fällen besondere Projektereignisse nachweisen. Also liegt die Vermutung nahe, dass bestimmte Ereignisse im Verlauf eines Softwareprojektes eine Abweichung der Kompelxitätsmetriken von den Aufwandsabschätzungen herbeiführen. Dieser Zusammenhang könnte in einer weiteren Analyse untersucht werden.

Zur Untersuchung der Analyseergebnisse wurde von einem Experten der DXC Technologies bereits eine sogenannte Umgebungsparameteranalyse skizziert. Diese wird als mögliche Weiterentwicklung dieser Arbeit im Folgenden beschrieben.

Im Rahmen der Umgebungsparameteranalyse werden als erster Schritt Fokus Punkte für die weitere Analyse ausgesucht. Das sollen Zeitintervalle seien, an denen die Codekomplexitätsmetriken über einen bestimmten Schwellenwert hinweg von den Aufwandsabschätzungen abweichen. Für das Alstonii Projekt ist die Auswahl dieser Zeitintervalle beispielhaft in Abbildung TODO skizziert.

TODO Grafik

Durch langjährige Erfahrung und Befragung von Projektbeteiligten konnten Umgebungsvariablen identifiziert werden, welche die Korrelation beeinflussen. Diese werden in drei Überkategorien gegliedert: Menschliche Faktoren (People), Prozessfaktoren (Process) und Technologiefaktoren (Technology). Zu jeder Hauptkategorie konnten Vier Messbereiche identifiziert werden. Diese werden mit unterschiedlichen quantitativen Indikatoren objektiv durch Befragung erfasst.

TODO Grafik

In einem dritten Schritt werden diese zwei Informationsstränge in Verbindung zueinander gesetzt. So sollen potenzielle Kausalitäten erkannt werden. Ein Umgebungsparameter könnte z.B. zu einer Abweichung der Komplexitätsmetriken von den Aufwandsabschätzungen führen. Der so generierte Kausalitätsgraph ist beispielhaft in Abbildung TODO skizziert.

TODO Grafik

In einem letzten Schritt soll versucht werden, aus den Kausalitäten Verbesserungsempfehlungen für das Management des Projektes abzuleiten. So könnte z.B. eine Änderung in der Arbeitsmethode (Prozessfaktor) eine Produktivitätserhöhung herbeiführen.

Insgesamt könnte mit der hier in Aussicht gestellten Arbeit ein System zur kontinuierlichen Verbesserung der Entwicklungstechnik in agilen Projekten geschaffen werden.

Anhang

Anhangverzeichnis

Anhang 1 So funktioniert's 63
Anhang 1/1 Wieder mal eine Abbildung 63
Anhang 1/2 Etwas Source Code 63
Anhang 2 Release Notes 63
Anhang 2/1 Änderungen in Version 1.1 63
Anhang 2/2 Änderungen in Version 1.2 64
Anhang 2/3 Änderungen in Version 1.3 65
Anhang 2/4 Änderungen in Version 1.4 67
Anhang 2/5 Änderungen in Version 1.5 67
Anhang 2/6 Änderungen in Version 1.6 68
Anhang 2/7 Änderungen in Version 1.7 70
Anhang 2/8 Änderungen in Version 1.8 70

Anhang 1: So funktioniert's

Um den Anforderungen der Zitierrichtlinien nachzukommen, wird das Paket `tocloft` verwendet. Jeder Anhang wird mit dem (neu definierten) Befehl **anhang**{Bezeichnung} begonnen, der insbesondere dafür sorgt, dass ein Eintrag im Anhangsverzeichnis erzeugt wird. Manchmal ist es wünschenswert, auch einen Anhang noch weiter zu unterteilen. Hierfür wurde der Befehl **anhangteil**{Bezeichnung} definiert.

In Anhang 1/1 finden Sie eine bekannte Abbildung und etwas Source Code in Anhang 1/2.

Anhang 1/1: Wieder mal eine Abbildung

Abb. 1: Mal wieder das DHBW-Logo.

Anhang 1/2: Etwas Source Code

Anhang 2: Release Notes

Anhang 2/1: Änderungen in Version 1.1

In Version 1.1 sind einige Rückmeldungen, die nach der Einführungsvorlesung am 6.2.2015 oder nach Veröffentlichung der Vorlage in Moodle eingegangen sind, berücksichtigt worden. Korrekturen sind mit „(Fix)“ gekennzeichnet.

- `latex-vorlage.tex`
  - (Fix) Abkürzungsverzeichnis wird vor Abbildungsverzeichnis platziert
  - (Fix) Abbildungs- und Tabellenverzeichnis in Inhaltsverzeichnis aufgenommen
  - (Fix) Quellenverzeichnis wird nun ohne Kapitelnummer dargestellt

  - eingebundene Dateien in Unterverzeichnissen `includes` bzw. `graphics`
  - Beispiel-Anhang (Datei `anhang.tex`) mit Erklärungen wurde eingebunden
- `_dhbw_praeambel.tex`
  - (Fix) das Paket hyperref wird nach biblatex eingebunden, um ein Problem mit der Verlinkung der Fußnoten im PDF zu beheben
  - (Fix) Fußnoten gemäß der Richtlinien fortlaufend nummeriert und nicht pro Kapitel
  - Einstellungen hinzugefügt, um Anhangsverzeichnis zu ermöglichen
  - bessere Kompatibilität zwischen KOMA-Script (scrreprt) und anderen Paketen mittels scrhack
- `_dhbw_biblatex-config.tex`
  - (Fix) keine Abschnittsnummern für einzelne Verzeichnisse im Quellenverzeichnis
- `abbildungen_und_tabellen.tex`
  - Erklärung, wie eine Fußnote/ein Zitat bei einer Abbildung zu erstellen ist
- `abkuerzungen.tex`
  - Abkürzungsverzeichnis wird im Inhaltsverzeichnis aufgeführt
- `abstract.tex`, `anhang.tex`, `einleitung.tex`
  - Erklärungen im Text ergänzt
- `deckblatt.tex`
  - Meta-Daten (Autor, Titel) für die generierte PDF-Datei lassen sich nun festlegen

Anhang 2/2: Änderungen in Version 1.2

Über das Forum in Moodle sind einige Rückmeldungen eingegangen – vielen Dank an alle, die dazu beigetragen haben. In der Version 1.2 wurden folgende Änderungen vorgenommen, wobei Korrekturen wieder mit „(Fix)“ gekennzeichnet sind:

- `latex-vorlage.tex` (Hauptdokument)
  - (Fix) Zeile 19: Seitenzahlen zu Beginn mit römischen *Groß*buchstaben nummeriert
- `_dhbw_praeambel.tex`
  - Zeile 39/40: Unterstützung für „ebenda“
  - Zeile 46–68: zweite Gliederungsebene für Anhänge ermöglicht

  - (Fix) Zeile 70–73: Abbildungen und Tabellen: Zähler fortlaufend, kein Rücksetzen zu Kapitelbeginn (Paket `chngcntr` anstelle von Paket `remreset`)

- `_dhbw_biblatex-config.tex`
  - (Fix) bei Quellen mit Herausgeber, aber ohne Autor wird der Name des Herausgebers im Verzeichnis fett gedruckt
  - Unterstützung für „ebenda“

- `abkuerzungen.tex`
  - Bemerkungen zur fortgeschrittenen Nutzung des `acronym`-Pakets eingefügt

- `einleitung.tex`
  - Abschnitt 1.3 zu Einstellungen ergänzt
  - Abschnitt 1.5 zu Fehlerbehebungen eingefügt

- `text-mit-zitaten.tex`
  - Abschnitt 3.1 eingefügt, Erläuterungen zum Zitieren mit „vgl.“ und „ebenda“.
  - Abschnitt 3.2: Beispiele ergänzt
  - Hinweis zu Jahreszahlen bei Online-Quellen

- `anhang.tex`
  - Erläuterungen zur zweiten Gliederungsebene

- `literatur-datenbank.bib`
  - weitere Beispiele für Quellen

Anhang 2/3: Änderungen in Version 1.3

Durch die ab 1/2016 geltenden Änderungen der Zitierrichtlinien des Studiengangs waren einige kleinere Anpassungen der Vorlage erforderlich, die nachfolgend beschrieben sind. Bei dieser Gelegenheit ebenfalls erfolgte Korrekturen sind wieder mit „(Fix)“ gekennzeichnet:

- `latex-vorlage.tex` (Hauptdokument)
  - Hinweis auf Option doppelseitiger Druck entfernt
  - Schriftgröße der Kapitelüberschriften verkleinert
  - (Fix) Kopf- und Fußzeilen werden nun korrekt angezeigt für erste Seite eines Kapitels und auch Quellenverzeichnisse

- `_dhbw_praeambel.tex`
  - Angabe des unteren Rands für Seitenzahl, da diese nun unten rechts steht
  - Unterstützung für „ebenda“ entfernt
  - (Fix) Präfixe wie „von“ im Namen eines Autors werden berücksichtigt
  - Anpassung der Abstände bei Kapitelüberschriften
  - Kopf- und Fußzeile für Verzeichnisse nun in `_dhbw_kopfzeilen.tex` definiert
- `deckblatt.tex`
  - Schriftgröße des Titels vergrößert
  - Befehl `\typMeinerArbeit` eingeführt, um Typ auszuwählen
  - Festlegung des Themas (für ehrenwörtliche Erklärung) mit Befehl `\themaMeinerArbeit`
  - Darstellung der Angabe des Betreuers in der Ausbildungsstätte angepasst
  - Formulierung des Sperrvermerks angepasst
- `_dhbw_erklaerung.tex`
  - Formulierung angepasst an geänderte Prüfungsordnung
  - Typ und Thema der Arbeit werden automatisch eingefügt
- `_dhbw_kopfzeilen.tex`
  - Seitennummern stehen jetzt unten rechts
  - (Fix) Kopf- und Fußzeile werden nun korrekt angezeigt in Verzeichnissen und dem Anhang
- `_dhbw_biblatex-config.tex`
  - Anpassung des Zitierstils auf die ab 1/2016 geltenden Regelungen
  - Vorkehrungen für Eindeutigkeit (Hinzufügen abgekürzter oder nötigenfalls ausgeschriebener Vorname) bei Übereinstimmung von Name und Jahreszahl
- `einleitung.tex`
  - Abschnitt 1.3 zu Einstellungen grundlegend überarbeitet
  - Abschnitt 1.5.2 zur Kontrolle der Seitenränder eingefügt
- `text-mit-zitaten.tex`
  - Abschnitt 3.1: Hinweise zu „ebenda“ entfernt

  - Abschnitt 3.2: Beispiele zur Eindeutigkeit des Zitats ergänzt
  - Abschnitt 3.3: Hinweise für E-Journals/E-Books ergänzt

- `anhang.tex`
  - (Fix) Befehl `\spezialkopfzeile` aufgenommen, damit in Kopfzeile das Wort „Anhang“ angezeigt wird
  - diese Release Notes wurden in eine eigene Datei verschoben

- `release_notes.tex`
  - s.o.

- `literatur-datenbank.bib`
  - weitere Beispiele für Quellen

Anhang 2/4: Änderungen in Version 1.4

Durch nicht abwärtskompatible Änderungen beim Versionswechsel von Biblatex 3.2 zu 3.3 sind einige Änderungen notwendig geworden.[212] Die vorliegende Version 1.4 wurde erfolgreich mit MikTeX gestestet (portable Version 2.9.6361 vom 3.6.2017, unter Verwendung von Biblatex 3.7).

- `_dhbw_biblatex-config.tex`
  - Anpassung der `\usebibmacro`-Befehle

- `_dhbw_authoryear.bbx`
  - Änderung von `\printdateextralabel` zu `\printlabeldateextra`

Anhang 2/5: Änderungen in Version 1.5

Für den Test dieser Version auf einem Windows-System wurde wieder die portable Version von MiKTeX (2.9.6521 vom 10.11.2017) verwendet.[213] Da in diesem Paket leider die Versionen von Biblatex (3.10) und Biber (2.7) inkompatibel sind, ist es erforderlich, die Datei `biber.exe` im Verzeichnis `texmfs\install\miktex\bin\` durch die aktuelle Version 2.10 vom 20.12.2017[214] zu ersetzen. Im Editor TeXworks verwendet man dann zum Übersetzen des LaTeX-Sourcecodes Typeset/pdfLaTeX bzw. Typeset/Biber.

Korrekturen sind wieder mit „(Fix)“ gekennzeichnet.

- `latex-vorlage.tex` (Hauptdokument)

[212] Diese basieren auf Vorschlägen von Yannik Ehlert – vielen Dank dafür!
[213] `http://miktex.org/portable`
[214] `https://sourceforge.net/projects/biblatex-biber/files/biblatex-biber/current/binaries/Windows/`

  - Nach der Änderung der Zitierrichtlinien gibt es nun kein separates Verzeichnis mehr für Internet- und Intranetquellen.
  - Option `notkeyword=ausblenden` bei `\printbibligraphy` sorgt dafür, dass Sekundärliteratur korrekt zitiert wird.

- `_dhbw_praembel.tex`
  - (Fix) Die Bezeichnung geschachtelter Anhänge wurde auf das in den Zitierrichtlinien geforderte Format „Anhang 2/1“ angepasst (Befehl `\anhangteil`).

- `einleitung.tex`
  - Hinweis zum Ausblenden der farbigen Links im PDF hinzugefügt

- `text-mit-zitaten.tex`
  - Abschnitt 3.4 aktualisiert nach Wegfall des separaten Verzeichnisses für Internet- und Intranetquellen
  - Abschnitt zum Zitieren von Sekundärliteratur hinzugefügt

Anhang 2/6: Änderungen in Version 1.6

Diese Version wurde auf einem Windows-System erfolgreich mit der portablen Version von MiKTeX (2.9.6621 vom 18.02.2018) getestet.[215]

Korrekturen sind wieder mit „(Fix)“ gekennzeichnet.

[215] Vielen Dank an Florian Eichin für seine wertvollen Anmerkungen.

- `latex-vorlage.tex` (Hauptdokument)
  - (Fix) An einer Stelle gab es in Version 1.5 (Internetquellen nicht mehr separat) noch ein Überbleibsel von Version 1.4 (Internetquellen separat), dies wurde korrigiert.
  - (Fix) Im Inhaltsverzeichnis war die Verlinkung des Abbildungs- und Tabellenverzeichnisses nicht ganz korrekt.
  - Mit den Befehlen `\literaturverzeichnis` bzw. `\literaturUndQuellenverzeichnis` kann bequem die Erstellung der Quellenverzeichnisse gesteuert werden, abhängig davon, ob es ein Gesprächsverzeichnis gibt oder nicht.
- `_dhbw_praembel.tex`
  - Einrückungen für Abbildungs-, Tabellen- und Anhangverzeichnis angepasst
  - Abkürzungen „Abb.“ und „Tab.“ für Abbildungen bzw. Tabellen
- `_dhbw_biblatex-config.tex`
  - Befehle `\literaturverzeichnis` und `\literaturUndGespraechsverzeichnis` definiert
  - Befehl `\footcitePrimaerSekundaer` definiert
- `_dhbw_erklaerung.tex`
  - Eintrag als „Erklärung“ (statt „Ehrenwörtliche Erklärung“) ins Inhaltsverzeichnis
- `einleitung.tex`
  - Bezeichnung „Erklärung“ statt „Ehrenwörtliche Erklärung“
  - Erläuterung von `\literaturverzeichnis` und `\literaturUndGespraechsverzeichnis`
  - Hinweis auf Notwendigkeit von Updates bei MikTeX Portable
- `text_mit_zitaten.tex`
  - Erläuterungen zu Befehl `\footcitePrimaerSekundaer` ergänzt
- `anhang.tex`
  - Befehl `\abstaendeanhangverzeichnis` für Anpassung Einrückung ergänzt
- `literatur-datenbank.bib`
  - Eintrag ergänzt

Anhang 2/7: Änderungen in Version 1.7

Diese Version wurde auf einem Windows-System erfolgreich mit der portablen Version von MiKTeX (2.9.6942 vom 04.01.2019) getestet.

Korrekturen sind wieder mit „(Fix)“ gekennzeichnet.

- `_dhbw-authoryear.bbx`
  - Da `labeldate` in Biblatex nicht mehr unterstützt wird, erfolgte eine Umbenennung in `labeldateparts`.[216]
- `_dhbw_biblatex-config.tex`
  - (Fix) Es wurde das Problem behoben, dass im Literaturverzeichnis bei bestimmten Eintragstypen der Titel in Anführungszeichen steht.[217]

Anhang 2/8: Änderungen in Version 1.8

Diese Version wurde auf einem Windows-System erfolgreich mit der portablen Version von MiKTeX (2.9.6942 vom 04.01.2019) getestet.

Die Aktualisierungen in der Vorlage spiegeln zum Einen die Änderungen in den Zitierrichtlinien wieder. Zum Anderen wurden einige studentische Vorschläge aufgegriffen, um die Nutzung der Vorlage zu erleichtern.[218]

- `latex_vorlage.tex` (Hauptdokument)
  - Es wird nun davon ausgegangen, dass die zur Vorlage gehörenden Dateien in einem eigenen Verzeichnis (`template`) liegen.
  - Stellenweise wurden Erläuterungen als Kommentare hinzugefügt.
- `_dhbw_biblatex-config.tex`
  - Code, der mehrere Quellenverzeichnisse unterstützt, wurde entfernt.
  - Ein zu großer Abstand nach Zitaten von Sekundärliteratur wurde korrigiert.
- `_dhbw_erklaerung.bbx`
  - Gemäß der Anforderung in den Zitierrichtlinien wird die Erklärung nicht ins Inhaltsverzeichnis aufgenommen und nicht mit einer Seitenzahl versehen.

[216] vgl. `https://github.com/semprag/biblatex-sp-unified/issues/23`
[217] Danke an Florian Eichin für seinen Hinweis.
[218] Danke an Bjarne Koll, Tobias Schwarz und Lars Ungerathen für ihre Anregungen.

- `_dhbw_praeambel.bbx`
  - Gemäß der Anforderung in den Zitierrichtlinien werden im Literaturverzeichnis alle Autor/innen eines Werks angegeben.
- `abstract.tex`
  - Hinweis auf LaTeX-Spickzettel hinzugefügt.
- `deckblatt.tex`
  - Vorname, Name, Titel der Arbeit sind nur zu Beginn einzutragen und werden dann an den entsprechenden Stellen automatisch ergänzt.
  - Hervorhebung, dass Angaben zum Unternehmen sowie den Betreuer/innen zu ergänzen sind.
  - Wortlaut des Vertraulichkeitsvermerks wurde an die aktuelle Fassung in der Studien- und Prüfungsordnung angepasst.
- `einleitung.tex`
  - Ein eigenständiges Gesprächsverzeichnis als Teil des Quellenverzeichnisses ist in den Zitierrichtlinien nicht mehr vorgesehen, die entsprechenden Hinweise wurden entfernt.
  - Ein alter Hinweis auf die Darstellung von Links im Verzeichnis der Internetquellen wurde entfernt, da es ein solches eigenständiges Verzeichnis nicht mehr gibt.
- `text_mit_zitaten.tex`
  - Es wird nun erläutert, wie zwei Quellenangaben unmittelbar nebeneinander dargestellt werden können.
  - Erklärungen, die von mehreren Quellenverzeichnissen ausgegangen sind, wurden entfernt.
- `literatur-datenbank.bib`
  - Gespräch wurde entfernt, da dieses nicht mehr im Quellenverzeichnis aufgeführt werden soll.

Literaturverzeichnis

About GitLab | GitLab (2022). URL: https://about.gitlab.com/company/,%20https://about.gitlab.com/company/ (Abruf: 04. 05. 2022).

Alenezi, M./Almustafa, K. (2015): Empirical Analysis of the Complexity Evolution in Open-Source Software Systems. In: *International Journal of Hybrid Information Technology* 8.2, S. 257–266. ISSN: 17389968. DOI: 10.14257/ijhit.2015.8.2.24. URL: http://gvpress.com/journals/IJHIT/vol8_no2/24.pdf (Abruf: 17. 03. 2022).

Atlassian (2022): What Is Agile? Atlassian. URL: https://www.atlassian.com/agile (Abruf: 22. 04. 2022).

Augsten, S. (2022): Was sind Softwaremetriken? URL: https://www.dev-insider.de/was-sind-softwaremetriken-a-813487/ (Abruf: 01. 05. 2022).

Benbasat, I./Goldstein, D. K./Mead, M. (1987): The Case Research Strategy in Studies of Information Systems. In: *Misquarterly MIS Quarterly* 11.3, S. 369–386. ISSN: 0276-7783.

Brückler, F. M. (2018): Geschichte der Mathematik kompakt: das Wichtigste aus Analysis, Wahrscheinlichkeitstheorie, angewandter Mathematik, Topologie und Mengenlehre. Berlin [Heidelberg]: Springer Spektrum. 158 S. ISBN: 978-3-662-55573-6. DOI: 10.1007/978-3-662-55574-3.

Cohn, M. (2004): User Stories Applied: For Agile Software Development. Addison-Wesley Signature Series. Boston: Addison-Wesley. 268 S. ISBN: 978-0-321-20568-1.

Cohn, M./Martin, R. C. (2006): Agile Estimating and Planning. Robert C. Martin Series. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference. 330 S. ISBN: 978-0-13-147941-8.

Compare Repositories - Open Hub (2022). URL: https://www.openhub.net/repositories/compare (Abruf: 03. 05. 2022).

Complexity (2022). thoughtbot, inc. URL: https://github.com/thoughtbot/complexity (Abruf: 05. 05. 2022).

Dalton, J. (2019): Great Big Agile: An OS for Agile Leaders. Berkeley, CA: Apress. ISBN: 978-1-4842-4205-6 978-1-4842-4206-3. DOI: 10.1007/978-1-4842-4206-3. URL: http://link.springer.com/10.1007/978-1-4842-4206-3 (Abruf: 22. 04. 2022).

Dubé/Paré (2003): Rigor in Information Systems Positivist Case Research: Current Practices, Trends, and Recommendations. In: *MIS Quarterly* 27.4, S. 597. ISSN: 02767783. DOI: 10.2307/30036550. JSTOR: 10.2307/30036550.

Dumke, R., Hrsg. **(1994)**: Theorie und Praxis der Softwaremessung. DUV Informatik. Wiesbaden: DUV, Dt. Univ.-Verl. 287 S. ISBN: 978-3-8244-2061-2.

Ebert, C., Hrsg. **(1996)**: Software-Metriken in der Praxis: Einführung und Anwendung von Software-Metriken in der industriellen Praxis. Berlin Heidelberg: Springer. 314 S. ISBN: 978-3-642-88195-4 978-3-540-60372-6.

Eisenhardt, K. M. (1989): Building Theories from Case Study Research. In: *The Academy of Management Review* 14.4, S. 532. ISSN: 03637425. DOI: 10.2307/258557. JSTOR: 258557.

Es6-Plato/Lib at Master · Deedubs/Es6-Plato (2022). GitHub. URL: https://github.com/deedubs/es6-plato (Abruf: 15. 04. 2022).

Fahrmeir, L./Heumann, C./Künstler, R./Pigeot, I./Tutz, G. (2016): Statistik: der Weg zur Datenanalyse. 8., überarbeitete und ergänzte Auflage. Springer-Lehrbuch. Berlin Heidelberg: Springer Spektrum. 581 S. ISBN: 978-3-662-50372-0 978-3-662-50371-3. DOI: 10.1007/978-3-662-50372-0.

Fenton, N. E./Pfleeger, S. L. (2003): Software Metrics: A Rigorous and Practical Approach. 2. ed., rev. print. Boston: PWS Publ. Comp. 638 S. ISBN: 978-0-534-95425-3.

Gil, J. (/Lalouche, G. (2016): When Do Software Complexity Metrics Mean Nothing? – When Examined out of Context. In: *The Journal of Object Technology* 15.1, 2:1. ISSN: 1660-1769. DOI: 10.5381/jot.2016.15.5.a2. URL: http://www.jot.fm/contents/issue_2016_01/article2.html (Abruf: 02. 05. 2022).

Gil, Y./Lalouche, G. (2017): On the Correlation between Size and Metric Validity. In: *Empirical Software Engineering* 22.5, S. 2585–2611. ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-017-9513-5. URL: http://link.springer.com/10.1007/s10664-017-9513-5 (Abruf: 02. 05. 2022).

Girtler, R. (2001): Methoden der Feldforschung. 4., völlig neu bearb. Aufl. UTB Soziologie 2257. Wien Köln Weimar: Böhlau. 198 S. ISBN: 978-3-205-99283-7 978-3-8252-2257-4.

GitLab 14 Delivers Modern DevOps in One Platform (2021). DevPro Journal. URL: https://www.devprojournal.com/news/gitlab-acquires-unreview-to-expand-its-open-devops-platform-with-machine-learning-capabilities-2/ (Abruf: 04. 05. 2022).

GitLab Communication (2022). GitLab. URL: https://about.gitlab.com/handbook/communication/ (Abruf: 04. 05. 2022).

GitLab Values (2022). GitLab. URL: https://about.gitlab.com/handbook/values/ (Abruf: 04. 05. 2022).

GitLab.Org / GitLab · GitLab (2022). GitLab. URL: https://gitlab.com/gitlab-org/gitlab (Abruf: 04. 05. 2022).

Göthlich, S. E. (2003): Fallstudien als Forschungsmethode: Plädoyer für einen Methodenpluralismus in der deutschen betriebswirtschaftlichen Forschung. Working Paper 578. Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel. URL: https://www.econstor.eu/handle/10419/147639 (Abruf: 06. 04. 2022).

Halstead, M. H. (1979): Elements of Software Science. 3. print. Operating and Programming Systems Series 2. New York: North Holland. 127 S. ISBN: 978-0-444-00205-1 978-0-444-00215-0.

Heinrich, L. J. (2005): Forschungsmethodik einer Integrationsdisziplin: Ein Beitrag zur Geschichte der Wirtschaftsinformatik. In: *NTM International Journal of History and Ethics of Natural Sciences, Technology and Medicine* 13.2, S. 104–117. ISSN: 0036-6978, 1420-9144. DOI: 10.1007/s00048-005-0211-9. URL: http://link.springer.com/10.1007/s00048-005-0211-9 (Abruf: 05. 04. 2022).

Hindle, A./Godfrey, M./Holt, R. (2008): Reading Beside the Lines: Indentation as a Proxy for Complexity Metric. In: *2008 16th IEEE International Conference on Program Comprehension*. 2008 IEEE 16th International Conference on Program Comprehension. Amsterdam: IEEE,

S. 133–142. ISBN: 978-0-7695-3176-2. DOI: 10.1109/ICPC.2008.13. URL: http://ieeexplore.ieee.org/document/4556125/ (Abruf: 02. 05. 2022).

Hindle, Abram/Godfrey, M. W./Holt, R. C. (2009): Reading beside the Lines: Using Indentation to Rank Revisions by Complexity. In: *Science of Computer Programming* 74.7, S. 414–429. ISSN: 01676423. DOI: 10.1016/j.scico.2009.02.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167642309000379 (Abruf: 01. 05. 2022).

Hoffmann, D. W. (2013): Software-Qualität. eXamen.press. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-35699-5 978-3-642-35700-8. DOI: 10.1007/978-3-642-35700-8. URL: http://link.springer.com/10.1007/978-3-642-35700-8 (Abruf: 05. 03. 2022).

@lasthash IEEE Standard for a Software Quality Metrics Methodology **(2022)**. IEEE. DOI: 10.1109/IEEESTD.1998.243394. URL: http://ieeexplore.ieee.org/document/749159/ (Abruf: 01. 05. 2022).

@lasthash IEEE Standard Glossary of Software Engineering Terminology **(2022)**. IEEE. DOI: 10.1109/IEEESTD.1990.101064. URL: http://ieeexplore.ieee.org/document/159342/ (Abruf: 19. 03. 2022).

Interview Mit Einem Entwickler Der inGRID Anwendung (2022). Unter Mitarb. von Entwickler.

Interview Mit Einer Mitarbeiterin Des DIL (2022).

Interview Mit Stakeholdern Des Englemanii Projektes (2022). Unter Mitarb. von Stakeholdern.

Interview Mit Vertretern Der Abteilung (2022).

Issues · GitLab.Org / GitLab · GitLab (2022). GitLab. URL: https://gitlab.com/gitlab-org/gitlab/-/issues (Abruf: 04. 05. 2022).

Jay, G./Hale, J. E./Smith, R. K./Hale, D./Kraft, N. A./Ward, C. (2009): Cyclomatic Complexity and Lines of Code: Empirical Evidence of a Stable Linear Relationship. In: *Journal of Software Engineering and Applications* 02.03, S. 137–143. ISSN: 1945-3116, 1945-3124. DOI: 10.4236/jsea.2009.23020. URL: http://www.scirp.org/journal/doi.aspx?DOI=10.4236/jsea.2009.23020 (Abruf: 17. 03. 2022).

Jones, C. (2008): Applied Software Measurement: Global Analysis of Productivity and Quality. 3rd ed. New York: McGraw-Hill. 662 S. ISBN: 978-0-07-150244-3.

Kemerer, C. F. (1987): An Empirical Validation of Software Cost Estimation Models. In: *Communications of the ACM* 30.5, S. 416–429. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/22899.22906. URL: https://dl.acm.org/doi/10.1145/22899.22906 (Abruf: 02. 12. 2021).

Landman, D./Serebrenik, A./Bouwers, E./Vinju, J. J. (2016): Empirical Analysis of the Relationship between CC and SLOC in a Large Corpus of Java Methods and C Functions: Empirical Analysis of the Relationship between CC and SLOC. In: *Journal of Software: Evolution and Process* 28.7, S. 589–618. ISSN: 20477473. DOI: 10.1002/smr.1760. URL: https://onlinelibrary.wiley.com/doi/10.1002/smr.1760 (Abruf: 02. 05. 2022).

Liggesmeyer, P. (2009): Software-Qualität: Testen, Analysieren und Verifizieren von Software. 2. Aufl. Heidelberg: Spektrum Akad. Verl. 526 S. ISBN: 978-3-8274-2056-5.

Lizard Python Module (o.J.). URL: https://pypi.org/project/lizard/.

Ltd, A. P. (2022): Use Advanced Search with Jira Query Language (JQL) | Jira Service Management Cloud. Atlassian Support. URL: https://support.atlassian.com/jira-

service-management-cloud/docs/use-advanced-search-with-jira-query-language-jql/ (Abruf: 06. 05. 2022).

McCabe, T. (1976): A Complexity Measure. In: *IEEE Transactions on Software Engineering* SE-2.4, S. 308–320. ISSN: 0098-5589. DOI: 10.1109/TSE.1976.233837. URL: http://ieeexplore.ieee.org/document/1702388/ (Abruf: 18. 12. 2021).

Mills, A. J./Durepos, G./Wiebe, E. (2010): Encyclopedia of Case Study Research. Los Angeles [Calif.]; London: SAGE. ISBN: 978-1-4129-5739-7 978-1-4522-6572-8 978-1-84972-768-6. URL: http://methods.sagepub.com/reference/encyc-of-case-study-research (Abruf: 08. 04. 2022).

Pandas - Python Data Analysis Library (2022). URL: https://pandas.pydata.org/ (Abruf: 06. 05. 2022).

Pandas.DataFrame.Corr — Pandas 1.4.2 Documentation (2022). URL: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html (Abruf: 03. 05. 2022).

Pandas.DataFrame.Cumsum — Pandas 1.4.2 Documentation (2022). URL: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.cumsum.html (Abruf: 03. 05. 2022).

Pandas.DataFrame.Mean — Pandas 1.4.2 Documentation (2022). URL: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.mean.html (Abruf: 03. 05. 2022).

Pandas.DataFrame.Merge — Pandas 1.4.2 Documentation (2022). URL: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html (Abruf: 03. 05. 2022).

Räcke, H. (2017): „Kontrollflussdiagramme“ (München). URL: http://www14.in.tum.de/lehre/2016WS/info1/split/sec-Kontrollflussdiagramme-twoscreen.pdf.

Revilla, M. A. (2007): Correlations between Internal Software Metrics and Software Dependability in a Large Population of Small C/C++ Programs. In: *The 18th IEEE International Symposium on Software Reliability (ISSRE '07)*. 2007 18th IEEE International Symposium on Software Reliability Engineering. Trollhattan: IEEE, S. 203–208. ISBN: 978-0-7695-3024-6. DOI: 10.1109/ISSRE.2007.12. URL: https://ieeexplore.ieee.org/document/4402211/ (Abruf: 16. 03. 2022).

Rubey, R. J./Hartwick, R. D. (1968): Quantitative Measurement of Program Quality. In: *Proceedings of the 1968 23rd ACM National Conference On* -. The 1968 23rd ACM National Conference. Not Known: ACM Press, S. 671–677. DOI: 10.1145/800186.810631. URL: http://portal.acm.org/citation.cfm?doid=800186.810631 (Abruf: 19. 03. 2022).

Rumreich, L. E./Kecskemety, K. M. (2019): Examining Software Design Projects in a First-Year Engineering Course Through Different Complexity Measures. In: *2019 IEEE Frontiers in Education Conference (FIE)*. 2019 IEEE Frontiers in Education Conference (FIE). Covington, KY, USA: IEEE, S. 1–5. ISBN: 978-1-72811-746-1. DOI: 10.1109/FIE43999.2019.9028569. URL: https://ieeexplore.ieee.org/document/9028569/ (Abruf: 14. 03. 2022).

Sarkar, D. ((2021): The Art of Effective Visualization of Multi-dimensional Data. Medium. URL: https://towardsdatascience.com/the-art-of-effective-visualization-of-multi-dimensional-data-6c7202990c57 (Abruf: 04. 05. 2022).

Sato, D./Bassi, D./Bravo, M./Goldman, A./Kon, F. (2006): Experiences Tracking Agile Projects: An Empirical Study. In: *Journal of the Brazilian Computer Society* 12.3, S. 45–

64. ISSN: 0104-6500, 1678-4804. DOI: 10.1007/BF03194495. URL: https://journal-bcs.springeropen.com/articles/10.1007/BF03194495 (Abruf: 17. 03. 2022).

Scaled Agile and GitLab (2022). GitLab. URL: https://about.gitlab.com/handbook/marketing/strategic-marketing/demo/executive-demo/ (Abruf: 04. 05. 2022).

Schwaber, K. (2004): Agile Project Management with Scrum. Redmond, Wash: Microsoft Press. 163 S. ISBN: 978-0-7356-1993-7.

Schwaber, K./Beedle, M. (2002): Agile Software Development with Scrum. Series in Agile Software Development. Upper Saddle River, NJ: Prentice Hall. 158 S. ISBN: 978-0-13-067634-4.

Simon, H. A. (2019): The Sciences of the Artificial. Third edition [2019 edition]. Cambridge, Massachusetts: The MIT Press. 231 S. ISBN: 978-0-262-53753-7.

Sneed, H. M./Seidl, R./Baumgartner, M. (2010): Software in Zahlen: die Vermessung von Applikationen. München: Hanser. 357 S. ISBN: 978-3-446-42175-2.

Stellman, A./Greene, J. (2014): Learning Agile. First edition. Beijing: O'Reilly. 397 S. ISBN: 978-1-4493-3192-4 978-1-4493-6384-0 978-1-4493-6382-6.

Technologies, D. (2022): Internes Dokument Zum Aufbau Der Digital NDA Application.

Weihman, K. (2021): *Multimetric*. Version 1.3.0. URL: https://github.com/priv-kweihmann/multimetric (Abruf: 03. 05. 2022).

Wilde, T./Hess, T. (2006): Methodenspektrum der Wirtschaftsinformatik: Überblick und Portfoliobildung. In: S. 20.

Yin, R. K. (2014): Case Study Research: Design and Methods. Fifth edition. Los Angeles: SAGE. 282 S. ISBN: 978-1-4522-4256-9.

Zuse, H. (1991): Software Complexity: Measures and Methods. Programming Complex Systems 4. Berlin ; New York: W. de Gruyter. 605 S. ISBN: 978-3-11-012226-8 978-0-89925-640-5.

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema: *Existiert eine Korrelation zwischen Storypoint-Aufwandsabschätzungen und Soft-warekomplexitätsmetriken? - Eine deskriptive Fallstudie sechs agiler Softwareprojekte* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

(Ort, Datum) (Unterschrift)