Existiert eine Korrelation zwischen Storypoint-Aufwandsabschätzungen und Softwarekomplexitätsmetriken?

Eine deskriptive Fallstudie sechs agiler Softwareprojekte

Bachelorarbeit

vorgelegt am 5. Mai 2022

Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik

Kurs WI2019I

von

Tim Struthoff

Betreuer in der Ausbildungsstätte:

DXC Technologies
(Titel) Andreas Jordan
⟨ Funktion der Betreuerin/des Betreuers ⟩

Unterschrift der Betreuerin/des Betreuers

DHBW Stuttgart:

Katja Sattler
Delivery Lead, Testing and Digital Assurance

LaTeX-Vorlage für Projekt-, Seminar- und Bachelorarbeiten

Bei dem vorliegenden Dokument handelt es sich um eine Vorlage, die für Projekt-, Seminar- und Bachelorarbeiten im Studiengang Wirtschaftsinformatik der DHBW Stuttgart verwendet werden kann.

Sie setzt die technischen Vorgaben der Zitierrichtlinien[1] des Studiengangs (Stand: 01/2020) um.

Hinweise: Bitte lesen Sie sich die Zitierrichtlinien unbedingt genau durch. Dieses Dokument ersetzt keine Anleitung oder Einführung in LaTeX, für die Nutzung sind daher gewisse Vorkenntnisse unerlässlich. Ein Einstieg in LaTeX ist aber weniger schwierig, als es vielleicht auf den ersten Blick scheint und lohnt sich für das Verfassen wissenschaftlicher Arbeiten in jedem Fall.[2] Als Hilfestellung beim Schreiben eines Dokuments habe ich einen zweiseitigen kompakten LaTeX-Spickzettel erstellt, der über Moodle verfügbar ist.

Ihre Rückmeldungen und Anregungen zu dieser Vorlage nehme ich gerne per E-Mail an die Adresse `tobias.straub@dhbw-stuttgart.de` entgegen.

— Prof. Dr. Tobias Straub

Versionshistorie

1.0	05.02.2015	erste Fassung
1.1	16.02.2015	siehe Anhang 2/1
1.2	20.04.2015	siehe Anhang 2/2
1.3	20.02.2016	siehe Anhang 2/3
1.4	24.07.2017	siehe Anhang 2/4
1.5	07.01.2018	siehe Anhang 2/5
1.6	07.04.2018	siehe Anhang 2/6
1.7	12.02.2019	siehe Anhang 2/7
1.8	10.02.2020	siehe Anhang 2/8

[1] Sie finden diese unter „Prüfungsleistungen“ im Studierendenportal (`https://studium.dhbw-stuttgart.de/winf/pruefungsleistungen/`).

[2] so auch `http://www.spiegel.de/netzwelt/tech/textsatz-keine-angst-vor-latex-a-549509.html`

Inhaltsverzeichnis

Abkürzungsverzeichnis

CRM Customer Relationship Management

DIL Digital Innovation Lab

Abbildungsverzeichnis

Tabellenverzeichnis

1 Cheatsheet

Text[3].

Referenz **??**

Abschnitten 1.1

`https://ctan.org/pkg/hyperref`

`_dhbw_`

```
_dhbw_biblatex-config.tex  (weitere Einstellung für Biblatex)
_dhbw_erklaerung.tex       (ehrenwörtliche Erklärung)
_dhbw_kopfzeilen.tex       (Kapitelname in Kopfzeilen)
_dhbw_praeambel.tex        (Einbindung der benötigten Pakete)
```

```
% HIER EDITIEREN:
```

1.1 Werkzeuge

1.1.1 Titel und Erklärung

Titel und Erklärung

Hinweis:

[3] Fußnote

2 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[4]. 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[5]. Im Interesse des Kunden sollten diese Kosten gesenkt werden[6].

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[7] und der durchschnittlichen Fehleranzahl korrelieren[8]. Also sei es sinnvoll, die Komplexität der Software permanent zu beobachten[9].

2.1 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.

[4] Interne quelle
[5] Quelle fehlt
[6] Interview Katja
[7] (Jones 2008 S. 64, 335, 627)
[8] (Jones 2008 S. 64, 503).
[9] (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.

2.2 Verwandte Arbeiten

Das Feld der Komplexitätsbestimmung von Software besteht bereits ähnlich lange wie die Softwareentwicklung selbst[10]. Erste Komplexitätsmaßzahlen, wie z.B. die zyklomatische Komplexität von McCabe[11] wurden in den sechziger und siebziger Jahren des zqwanzigsten Jahrhunderts entwickelt[12] [13]. 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[14] werden verschiedene Komplexitätsmaßen auf 15 Projekte einer Firma angewendet. Dabei kommen die Codezeilen basierten Maßen SLIM und COCOMO, sowie die nicht-Codezeilenbasierten Maßen ESTIMACS und Function Points zum Einsatz[15]. 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[16].

In rumreichExaminingSoftwareDesign2019 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[17]

In alenezi EmpiricalAnalysisComplexity2015 wird der generelle Verlauf von Softwarekomplexitätsmetriken untersucht. Dabei kann das sechste Lehmansche Gesetz bewiesen warden, nach dem die Komplexität einer Software über Zeit steigt[18].

[10] quelle
[11] McCabe, T. J. 1976
[12] Zuse, H. 1991, S. 25
[13] Rubey, R. J./Hartwick, R. D. 1968
[14] Kemerer, C. F. 1987
[15] Ebenda, S. 2
[16] Ebenda, S. 12
[17] (Rumreich and Kecskemety 2019:1)
[18] (Alenezi and Almustafa 2015:262)

2.3 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.

2.4 Methodisches Vorgehen

Als Abschlussarbeit eines Wirtschaftsinformatik (WI) Studiums wird sich diese Arbeit auch an den Methodiken der WI orientieren. Die WI ist in ihrer Erkenntnisgewinnung methodenpluralistisch aufgestellt[19]. Ihr instrumentales Portfolio beinhaltet sowohl Methodiken aus den Real-, den Formal- sowie den Ingenieurswissenschaften[20].

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[21].

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[22]. Es weist z.B. Vorgehen der Informationssystemsgestaltung, wie das Prototyping auf[23]. Dem gegenüber steht das behavioristische und verhaltenswissenschaftliche Paradigma. Nach diesem Paradigma wird in der WI das Verhalten und die Auswirkungen von bestehenden Informationssystemen untersucht[24].

[19] Wilde, T./Hess, T. o. J., S. 1
[20] Ebenda, S. 1
[21] Herrmann (1999)
[22] (Wilde/Hess S. 2).
[23] (Wilde/Hess S. 3 und Simon 1998)
[24] (vgl. Wilde / Hess 2006 S. 3)

In dieser Arbeit soll nach dem behavioristischen Paradigma eine Erkenntnis gewonnen werden[25]. 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[26] [27] [28]. 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[29]. Vielmehr soll eine unvollständige Theorie in Form einer Näherungsangabe als ceteris-paribus Hypothese aufgestellt werden[30].

Trotz ihres vergleichsweisen jungen Alters bietet die WI eine üppige Bandbreite an Methoden der Erkenntnisgewinnung[31]. Gerade die Methodik der Fallstudienforschung erlebt einen stetigen Anstieg in Popularität[32]. Insbesondere für eine beschreibende Forschung, wie sie in dieser Arbeit angestrebt wird, sei die Fallstudienmethodik geeignet[33]. 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[34]. 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 [35]. Der Aufbau der Fallstudie[36] wird in Kapitel 4 weiter beschrieben.

[25](vgl. Wilde / Hess 2006 S. 3).
[26](Zuse, 1991, p. 561)
[27](Zuse, 1991, p. 562)
[28]Nach BOWL83 sei eine Metrik dann validiert, wenn sie das misst, was sie angibt zu messen.
[29](Jones 2008:449)
[30](vgl. Wilde / Hess S. 3).
[31]heinrichForschungsmethodikIntegrationsdisziplinBeitrag2005 S. 113
[32](yinCaseStudyResearch2014 S. 22 /Michel et al., 2010 / David, 2006b, p. xxiii / David, 2006a / Mills, Durepos, & Wiebe, 2010a))
[33](dubeRigorInformationSystems2003 S. 607).
[34]gothlichFallstudienAlsForschungsmethode2003 S. 7
[35]gothlichFallstudienAlsForschungsmethode2003 S. 6
[36]gothlichFallstudienAlsForschungsmethode2003 S. 8ff

2.5 Aufbau der Arbeit

Der Aufbau dieser Arbeit orientiert sich an dem Aufbau ähnlicher Arbeiten[37] 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 und 2.2). In einem zweiten Schritt werden verschiedene Methoden zur Messung von Softwarekomplexität begründet ausgewählt und erklärt (Kapitel 2.4).

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).

[37] (Alenezi and Almustafa 2015:260), "5.1. Selected Systems"(Alenezi and Almustafa 2015:260), "Threats to Validity"(Alenezi and Almustafa 2015:264)

3 Softwarekomplexitätsmaße

Die Analyse der Softwarekomplexität ist ein Teil der statischen Sourcecode-Analyse. In der Code-Analyse wird im Nachhinein (a posterio[38]) anhand von statischen Analysen einer Stichprobe zu einem bestimmten Zeitpunkt[39] des Sourcecodes festgestellt, ob die Software vorher definierten Qualitätsanforderungen entspricht[40]. Diese Analyse kann sowohl automatisiert als auch manuell erfolgen und wird im Gegensatz zu Verfahren der konstruktiven Qualitätssicherung auf bereits vorhandene Software angewendet[41].

Als Teil der statischen Code-Analyse ist die Messung der Softwarekomplexität ein Teil der Qualitätssicherung der Software[42] [43]. Dabei sollen oft unsichtbare Eigenschaften der Software sichtbar und quantitativ messbar gemacht werden[44] [45]. 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[46] [47]. 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[48].

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[49].

3.1 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 IEEE Standard Glossar der Softwareentwicklung besteht *Software* aus den Computerprogrammen, Prozeduren und gegebenenfalls der Dokumentation und den Daten, die den Betrieb eines Computersystems betreffen[50]

[38] Hoffmann, D. W. 2013, S. 261
[39] (Ebert 1996: 86)
[40] Ebenda, S. 261
[41] (Hoffmann 2013:261)
[42] Quelle fehlt
[43] Hoffmann 2013:261
[44] Hoffmann 2013:261
[45] Zuse, 1991, p. S. 561)
[46] Hoffmann, D. W. 2013, S. 22f
[47] Liggesmeyer 2009:245
[48] Quelle fehlt
[49] Quelle fehlt
[50] IEEEStandardGlossary S. 66

Die Definition von *Komplexität* wird allgemein als schwierig erachtet[51] [52]. 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“[53]. 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[54]. 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[55].

Im Kontext der Softwaremetrie werden Metriken als ein Messystem bzw. ein Verfahren zum Quantifizieren von Eigenschaften von Software definiert[56] [57] [58] [59]. 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[60]. Ein Maß (measure) sei dahingegen die Konkrete Anwendung dieser Metrik[61]. 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.

3.2 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[62].

[51] Quelle fehlt
[52] Jones 2008:335, Jones 2008:627
[53] O. V. o. J., S. 18
[54] (fentonSoftwareMetricsRigorous2003 S. 258, Jones 2008:449).
[55] Zuse, H. 1991, S. 7ff
[56] dumkeTheorieUndPraxis1994 S. 35ff
[57] ebertSoftwareMetrikenPraxisEinfuhrung1996 S. 4ff
[58] augstenWasSindSoftwaremetriken
[59] IEEEStandardSoftware S. 2f
[60] IEEEStandardSoftware S. 3
[61] IEEEStandardSoftware S. 2
[62] Rumreich and Kecskemety 2019:2

Weiter werden auch die Komplexitätsmetriken im Speziellen angezweifelt: Viele Metriken vereinen mehrere, oft konfliktäre Messziele[63]. 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[64]. 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[65]. Heutzutage sind andere Programmiertechniken, wie z.B. die objekt-orientierte Programmierung verbreiteter. Diese neuen Programmiertechniken wurden jedoch in den klassischen Metriken nicht berücksichtigt[66]. 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[67].

Ferner ist auch der Zusammenhang zwischen Softwaremetriken und externen Softwareeigenschaften, wie z.B. der Fehleranfälligkeit umstritten[68]. 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[69]. Im Gegensatz dazu konnten in Revilla 2007 keine Zusammenhänge zwischen internen Softwaremetriken und externen Eigenschaften von Software festgestellt werden[70].

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.

3.3 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

[63] fentonSoftwareMetricsRigorous2003 S. 322
[64] Rumreich and Kecskemety 2019:2
[65] Hoffmann 2013:277
[66] Hoffmann 2013:277
[67] Hoffmann 2013:277
[68] Jones 2008:627
[69] Ebert 1996:65
[70] Revilla 2007:203,208

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[71]. Diese Position wird auch von satoExperiencesTrackingAgile2006 und alenezi EmpiricalAnalysisComplexity2015 bekräftigt. Die zyklomatische Komplexität von McCabe wird ebenfalls von Zuse als eine der bekanntesten Maßzahlen eingestuft [72]. Weiter bekräftigt fentonSoftwareMetricsRigorous2003, dass diese Maßzahl zum Messen der Softwarekomplexität geeignet sei[73]. Auch die vier Autoren Revilla[74], Jones[75], Sato[76] und Alenezi[77] 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[78]. Auch die Softwaremaßzahlen von Halstead werden in der Literatur häufig referenziert. Zuse und Revilla bekräftigen die Verbreitung dieser Maßzahlen[79]. Zusätzlich sagt Fenton aus, dass die Maßzahlen geeignet zum Messen der Softwarekomplexität seien[80].

Zusätzlich wird die Einrückungskomplexität betrachtet, da sie einen, im Gegensatz zu den anderen Maßzahlen methodisch sehr abweichenden Ansatz verfolgt[81]. 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)[82], die Anzahl an Kommentaren[83], die Größe der Klasse[84] und die Anzahl an Testcodezeilen[85] 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

[71] Zuse 91 S. 145
[72] Zuse 91 S. 145
[73] fentonSoftwareMetricsRigorous2003 S. 31
[74] Revilla 2007:203
[75] Jones 2008:S. 335, 627, 449
[76] satoExperiencesTrackingAgile2006
[77] alenezi EmpiricalAnalysisComplexity2015
[78] Jones 2008:S. 335, 627, 449
[79] Zuse 91 S. 145, Revilla 2007:203
[80] fentonSoftwareMetricsRigorous2003 S. 31
[81] Quelle fehlt
[82] satoExperiencesTrackingAgile2006
[83] Revilla 2007:203
[84] satoExperiencesTrackingAgile2006
[85] satoExperiencesTrackingAgile2006

nach Halstead[86]. Eine noch stärkere, linear-stabile Korrelation lässt sich zwischen der Anzahl logischer Code Zeilen und der zyklomatischen Komplexitätsmaßzahl nachweisen. Diese Korrelation konnte über Programmierer*innen, Programme, Sprachen und Programmierparadigma hinweg bewiesen werden[87].

Die vier Komplexitätsmetriken werden nun der Reihe nach genauer erläutert.

3.3.1 Logische Codezeilen

Als erste hier vorgestellte Metrik weist die Anzahl der logischen Codezeilen (SLOC) den geringsten Berechnungsaufwand auf. Sie wird teilweise auch als LOC bzw. Lines of Code bezeichnet[88]. Dabei werden die Zeilen an Sourcecode in der Software berechnet[89]. Die Metrik lässt sich somit ohne die Unterstützung komplexer Algorithmen berechnen und lässt sich auf nahezu alle Programmiersprachen anwenden[90]. Eine Ausnahme bilden hier grafische Programmiersprachen[91].

In ihrer Aussagekraft wird die Anzahl von Codezeilen allgemein als begrenzt eingestuft[92]. 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[93] 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 (Hoffmann 2013:264). Zusätzlich bestehen auch empirisch ermittelte Sprachfaktoren, die es ermöglichen, die Ergebnisse von LOC und NCSS Metriken vor ihrer Weiterverarbeitung zu gewichten (Hoffmann 2013:264).

3.3.2 Zyklomatische Komplexität

Die zyklomatische Komplexität wurde von Tom McCabe entwickelt und zuerst in seinem Aufsatz „A Complexity Metric“[94] veröffentlicht[95]. Die zyklomatische Komplexität gehört zu den Kontrollflussmetriken[96]. In der Praxis wird sie von vielen Firmen entwicklungsbegleitend kontinuierlich

[86] Jones 2008: 627
[87] Jones 2008:627, Jay et al 2009:137
[88] Hoffmann 2013:263
[89] Rumreich and Kecskemety 2019:2
[90] Hoffmann 2013:263
[91] Quelle fehlt
[92] Hoffmann 2013:264, Rumreich and Kecskemety 2019:2
[93] zuseSoftwareComplexityMeasures1991 S. 142
[94] (McCabe 76)
[95] (Sneed et al 2010:185).
[96] (Ebert 1996:88).

erhoben, um vor problematischen Programmkomponenten frühzeitig zu warnen. Dabei fließt sie oft auch in die Abnahmekriterien für Software ein[97].

Die zyklomatische Komplexität ist definiert als eine Maßzahl für die Kontrollflusskomplexität eines Programmes[98]. Sie misst die Anzahl von linear unabhängigen Pfaden auf dem Kontrollflussgraphen eines Programmes und liefert so ein normalisiertes Komplexitätsmaß[99].

Zur Berechnung der zyklomatischen Komplexität wird der Programmcode zunächst in einen Graphen aus Knoten und Kanten umgewandelt[100]. 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[101]. Der Umwandlungsprozess wird in Abbildung .. beschrieben

GRAFIK

Aus den Eigenschaften dieses Kontrollflussgraphen kann nun die zyklomatische Komplexität mit Mitteln der Graphentheorie[102] 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[103].

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[104]. 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.

[97] (Hoffmann 2013:275).
[98] (Rumreich and Kecskemety 2019:2, Jones 2008:335).
[99] (Rumreich and Kecskemety 2019:2 (Quelle 15 dort)).
[100] (Sneed et al 2010:185, Jones 2008:335).
[101] rackeKontrollflussdiagramme2017
[102] (Hoffmann 2013:273).
[103] (Hoffmann 2013:275f)
[104] (Jones 2008:336)

Zunächst wird allgemein kritisiert, dass die Maßzahl nur die Komplexität der Ablauflogik des Codes und nicht die Komplexität des gesamten Codes misst[105]. Interne Eigenschaften der Knoten[106] [107], die Komplexität im Datenfluss[108] sowie die Verschachtelung von Modulen werden ignoriert[109]. 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[110].

Insgesamt wird geschlussfolgert, dass die zyklomatische Komplexität nur mit Vorsicht anzuwenden sei und in Relation zu anderen Maßzahlen gesetzt werden müsse[111]. Genau aus diesem Grund wird sie in dieser Arbeit in den Kontext der agilen Aufwandsabschätzungen und in Relation zu anderen Maßzahlen gesetzt.

3.3.3 Halstead Metriken

Als dritte Komplexitätsmetrik werden die Halstead Metriken herangezogen. Bei diesen Metriken wendet der Autor Maurice Howard Halstead[112] Elemente der Kommunikationstheorie auf die Programmierung an[113].

Die Berechnung der Halstead Metriken wird im folgenden an einem Beispiel erklärt[114]. Das Beispiel behandelt folgenden Code:

main() (Programmiersprache C)

{

int countOne, countTwo, countThree, average;

scanf(“%d %d %d”, &countOne, &countTwo, &countThree);

average = (countOne + countTwo + countThree) / 3;

printf(“average = %d”, average);

}

Für die Berechnung seiner Metriken zählt Halstead zunächst die Grundelemente der Programmiersprache, nämlich die Operatoren und Operanden[115]. Operatoren führen Operationen auf

[105](Sneed et al 2010:186).
[106](Hoffmann 2013:273).
[107](Rumreich and Kecskemety 2019:2)
[108](Rumreich and Kecskemety 2019:2)
[109](Zuse S. 89ff).
[110](Sneed et al 2010:186).
[111](fentonSoftwareMetricsRigorous2003 S. 52).
[112]halsteadElementsSoftwareScience1979
[113](Sneed et al 2010:183)
[114](Sneed et al 2010:184)
[115](Sneed et al 2010:183), (Rumreich and Kecskemety 2019:2)

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, „ ;

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[116]. 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[117]. Eine zusätzliche Einschränkung sei, dass die Halstead Metriken die Berechnungskomplexität in den Vordergrund rücken und keine Aussage über den Kontrollfluss der Anwendung treffen[118]. Insgesamt seien sie also kritisch anzusehen[119].

3.3.4 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[120]. Die Komplexitätsmaßzahl ist im Vergleich zu den anderen hier behandelten Metriken verhältnismäßig neu, wurde aber bereits in Arbeiten von Lalouche et. al[121] [122]und Landman et. al [123] 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[124].

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[125]. 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

[116](Sneed et al 2010:185)
[117]Zuse, H. 1991, S. 142
[118](Rumreich and Kecskemety 2019:2, Quelle 16 dort
[119](Sneed et al 2010:185)
[120]hindleReadingLinesIndentation2008 S. 1
[121]gilWhenSoftwareComplexity2016 S. 10
[122]gilCorrelationSizeMetric2017 S. 7
[123]landmanEmpiricalAnalysisRelationship2016 S. 6
[124]hindleReadingLinesUsing2009 S. 1
[125]

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[126]. Diese Korrelation konnte von den Autoren in einer Studie von 278 populären Open Source Projekten erfolgreich nachgewiesen werden[127]. 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[128]. In den meisten Projekten entsprechen vier Leerzeichen bzw. ein Tab einem Einrückungslevel[129]. 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[130] [131].

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[132]. 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)

[126] hindleReadingLinesUsing2009 S. 2
[127] Quelle
[128] hindleReadingLinesUsing2009 S. 5
[129] hindleReadingLinesUsing2009 S. 9
[130] hindleReadingLinesUsing2009 S. 1
[131] hindleReadingLinesUsing2009 S. 2
[132] hindleReadingLinesUsing2009 S. 20f

- 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[133]. 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[134]. Eine zusätzliche Einschränkung sei, dass die Halstead Metriken die Berechnungskomplexität in den Vordergrund rücken und keine Aussage über den Kontrollfluss der Anwendung treffen[135]. Insgesamt seien sie also kritisch anzusehen[136].

[133](Sneed et al 2010:185)
[134]Zuse, H. 1991, S. 142
[135](Rumreich and Kecskemety 2019:2, Quelle 16 dort
[136](Sneed et al 2010:185)

3.3.5 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[137]. Die Komplexitätsmaßzahl ist im Vergleich zu den anderen hier behandelten Metriken verhältnismäßig neu, wurde aber bereits in Arbeiten von Lalouche et. al[138] [139]und Landman et. al [140] 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[141].

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[142]. 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[143]. Diese Korrelation konnte von den Autoren in einer Studie von 278 populären Open Source Projekten erfolgreich nachgewiesen werden[144]. 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

[137] hindleReadingLinesIndentation2008 S. 1
[138] gilWhenSoftwareComplexity2016 S. 10
[139] gilCorrelationSizeMetric2017 S. 7
[140] landmanEmpiricalAnalysisRelationship2016 S. 6
[141] hindleReadingLinesUsing2009 S. 1
[142]
[143] hindleReadingLinesUsing2009 S. 2
[144] Quelle

werden für jede Zeile berechnet. Anhand dieser Angaben wird auf der Ebene einer Datei berechnet, wie viele Leerzeichen bzw. Tabs einer logischen Einrückungsebene entsprechen[145]. In den meisten Projekten entsprechen vier Leerzeichen bzw. ein Tab einem Einrückungslevel[146]. 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[147] [148].

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[149].

[145] hindleReadingLinesUsing2009 S. 5
[146] hindleReadingLinesUsing2009 S. 9
[147] hindleReadingLinesUsing2009 S. 1
[148] hindleReadingLinesUsing2009 S. 2
[149] hindleReadingLinesUsing2009 S. 20f

4 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.

4.1 Die agile Arbeitsweise in der Softwareentwicklung

Als die agile Arbeitsweise wird eine Reihe von iterativen Methoden[150] 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[151].

Zusätzlich wird mit der agilen Arbeitsweise eine Mentalität beschrieben, bei der Planung, Design und Prozessoptimierung von dem gesamten Entwicklungsteam durchgeführt werden sollen[152].

4.2 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.

[150] atlassianWhatAgile
[151] stellmanLearningAgile2014 S. 2
[152] stellmanLearningAgile2014 S. 2

4.2.1 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[153] [154].

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[155]. 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[156]. Von den meisten Scrum-Teams wird eine feste Zahlenfolge zur Schätzung der Aufwände verwendet. Dabei ist die Fibonacci-Folge[157] besonders beliebt[158].

4.2.2 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[159] [160].

[153] schwaberAgileSoftwareDevelopment2002
[154] schwaberAgileProjectManagement2004
[155] cohnUserStoriesApplied2004
[156] Quelle fehlt
[157] erklären
[158] Quelle fehlt
[159] Quelle ergänzen cohnAgileEstimatingPlanning2006
[160] (daltonGreatBigAgile2019a S. 203).

5 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 (gothlichFallstudienAlsForschungsmethode2003 S. 8, millsEncyclopediaCaseStudy2010 S. 69f). 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 (millsEncyclopediaCaseStudy2010 S. 69f).

5.1 Forschungsfrage

Eine klare Forschungsfrage ist essenziell für die Fallstudienforschung, um die Problemstellung und Ziele des Handels aufzuzeigen (dubeRigorInformationSystems2003 S. 605, millsEncyclopediaCaseStudy2010 S. 70, dubeRigorInformationSystems2003 S. 607). In der Fallstudienforschung können Forschungsfragen explorativ, deskriptiv und explanativ sowie eine Kombination aller drei Arten sein (dubeRigorInformationSystems2003 S. 605).

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.

5.2 Hypothesen

Vor der Beantwortung der Forschungsfrage werden zunächst Hypothesen über mögliche Antworten aufgestellt (gothlichFallstudienAlsForschungsmethode2003 S. 8). Das Aufstellen von Hypothesen dient der Identifikation der weiteren Forschungsrichtung (dubeRigorInformationSystems2003 S. 607, millsEncyclopediaCaseStudy2010 S. 70).

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.

5.3 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 (dubeRigorInformationSystems2003 S. 610).

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 (millsEncyclopediaCaseStudy2010 S. 72f, yinCaseStudyResearch2014 S. 72). 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[161]. Als Grundvoraussetzung zur Auswahl der Fälle lässt sich sagen, dass ein möglichst umfassender Zugang zu Dokumenten und Artefakten des Falles gegeben sein sollte (millsEncyclopediaCaseStudy2010 S. 68). Für eine möglichst valide Endaussage sei eine Anzahl von vier bis zehn Fällen anzustreben (gothlichFallstudienAlsForschungsmethode2003 S. 9[162], dubeRigorInformationSystems2003 S. 609).

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[163]. 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.

[161] Quelle fehlt (Ausreißer und Extremfälle gerade an diesen Extremfällen gelegen, da sie die Bandbreite abstecken, innerhalb derer sich die Realität bewegt und relevante Phänomene in diesen Fällen am deutlichsten zu Tage treten.)

[162] Eisenhardt, Kathleen M. (1989): 'Building Theories from Case Study Research', in: Academy of Management Review, Vol. 14, No. 4, S. 532-550.

[163] GitLabGitLab

- 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.

5.4 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[164] [165] [166]. Die Auswahl und Beschreibung der Datenquellen wird in verschiedenster Literatur umfassend erläutert[167].

Für die Auswahl der Datenquellen kommen verschiedene Quellenformate in Frage: Im Generellen seien Dokumente jeglicher Art als Quelle zulässig[168]. Auch Archivdaten spielen eine wichtige Rolle. So wurde in einer Metastudie ermittelt, dass in ca. 64% aller beobachteten Fallstudien Archivdaten als Quelle eingesetzt wurden[169]. Hier seien insbesondere Zeitreihendaten von Interesse

[164] (dubeRigorInformationSystems2003 S. 612)
[165] 1. (p. 381) (dubeRigorInformationSystems2003 S. 614)
[166] Benbasat et al.'s (1987)
[167] (gothlichFallstudienAlsForschungsmethode2003 S. 10); Girtler, Roland (2001): Methoden der Feldforschung, 4., völlig neu bearbeitete Auflage, Wien, Köln, Weimar; Lueger, Manfred (2000): Grundlagen qualitativer Feldforschung, Me-thodologie, Organisierung, Materialanalyse, Wien; Piore, Michael J. (1979): 'Qualitative Research Techniques in Econom-ics', in: Administrative Science Quarterly, Vol. 24 (1979), No. 4, S. 560-569.
[168] (gothlichFallstudienAlsForschungsmethode2003 S.10)
[169] (dubeRigorInformationSystems2003 S. 614)

[170]. Weiter seien Interviews, Beobachtungen und Befragungen zulässige Datenquellen[171]. Auch sämtliche andere Artefakte der Fälle, sowohl in physischer als auch in digitaler Form können zu der Untersuchung hinzugezogen werden[172].

Zu diesen Primärdaten könnten auch noch Sekundärdaten, wie z.B. Interpretationen, Zusammenfassungen, Textanalysen, Statistiken und Berichte hinzugezogen werden (gothlichFallstudienAlsForschungsmethode2003 S. 11).

Für eine möglichst valide Endaussage sei eine Kombination der Quellen besonders sinnvoll (gothlichFallstudienAlsForschungsmethode2003 S. 10). Diese Verwendung mehrerer Quellen und auch mehrerer Forschungsmethoden kann als methodologische Triangulation verstanden werden (gothlichFallstudienAlsForschungsmethode2003 S. 10).

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 (gothlichFallstudienAlsForschungsmethode2003 S. 11).

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.

5.5 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 (yinCaseStudyResearch2014 S. 78, gothlichFallstudienAlsForschungsmethode2003 S. 11). Diese Reflektion bzw. Verbindung kann mit verschiedenen Analyseverfahren hergestellt werden. Dabei stehen unter anderem Mustervergleiche, Erklärungsgebilde, Zeitserienanalysen, logische Modelle und fallübergreifende Synthesen zur Verfügung (yinCaseStudyResearch2014 S. 78, gothlichFallstudienAlsForschungsmethode2003 S. 11). Wenn bekannt ist, dass einige oder alle Hypothesen eine Entwicklung über Zeit betrachten, könne zum Beispiel eine Zeitserienbetrachtung sinnvoll sein (yinCaseStudyResearch2014 S. 78, 225).

In dieser Arbeit wird die Komplexität der Software in Projekten betrachtet (siehe Unit of Analysis). Dabei sollen die Komplexitätsmetriken mit Aufwandsabschätzungen verglichen werden.

[170] i. (Benbasatet al. 1987; Eisenhardt1989). (dubeRigorInformationSystems2003 S. 612)
[171] (Benbasatet al. 1987; Eisenhardt1989). (dubeRigorIn-formationSys-tems2003 S. 612), Yin(1994)
[172] a. (gothlichFallstudienAlsForschungsmethode2003 S.10); (dubeRigorInformationSystems2003 S. 612); Yin(1994)

Dieser Vergleich bzw. diese Verbindung ist in Abbildung .. dargestellt und wird im Folgenden erläutert.

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[173]. 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 .. 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

[173] Quelle Git

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.

5.6 Kriterien zur Interpretation der Daten

Die in 4. 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 (gothlichFallstudienAlsForschungsmethode2003 S. 11?).

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 (yinCaseStudyResearch2014 S. 79, gothlichFallstudienAlsForschungsmethode2003 S. 11??) sowie Interviews (gothlichFallstudienAlsForschungsmethode2003 S. 11?). Mit diesen beiden Techniken werden die Analyseergebnisse aus 4.2.4 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[174].

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[175].

[174] (Jay et al 2009:140), hindleReadingLinesUsing2009
[175] bucklerGeschichteMathematikKompakt2018 S. 116

Die Ergebnisse können über ihren Betrag grob eingeordnet werden[176]:

- „schwache Korrelation“ r <= 0.5
- „mittlere Korrelation“ 0.5 <= r <= 0.8
- „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[177]. 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[178]. 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 (gothlichFallstudienAlsForschungsmethode2003 S. 12). Dabei birge eine zu detailreiche Analyse der Interviews die Gefahr der Überinterpretation. Wichtiger sei es, die Ergebnisse der Interviews kritisch zu reflektieren (gothlichFallstudienAlsForschungsmethode2003 S. 12). 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 (gothlichFallstudienAlsForschungsmethode2003 S. 11). 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

[176] fahrmeirStatistikWegZur2016 S. 130
[177] fahrmeirStatistikWegZur2016 S. 137ff
[178] fahrmeirStatistikWegZur2016 S. 133f

ist vorstellbar. Dabei kommen verschiedene Fallstudien zu verschiedenen Ergebnissen, welche aber theorieseitig erklärbar sind (gothlichFallstudienAlsForschungsmethode2003 S.11).

5.7 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.

5.7.1 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 4.2.4 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[179] vor. Diese Versionshistoriendaten sollen von der Untersuchungssoftware verarbeitet werden können.

- 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 CSV Datei vorgesehen:

TODO!TABLE

- Als zweiter Ausgabeparameter wird ein Graph des Verlaufes aller Maßzahlen im PNG- und 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[180] 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.

1. Aufbau der Untersuchungssoftware

[179] Git erklären
[180] Definition Heatmap

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[181].

b. **Plato**: Das NodeJS Commandline-Tool ES6-Plato[182] 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[183] wird die Einrückungskomplexität für alle Programmiersprachen berechnet

d. **MulitMetric**: Das MultiMetric Python Modul[184] 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[185] 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

5.7.2 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 4.2.4 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

[181] LizardPythonModule
[182] Es6platoLibMaster
[183] Complexity2022
[184] weihmanMultimetric2021
[185] (Hoffmann 2013:276)

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[186]. 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[187]. 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)

[186] Quelle fehlt
[187] `https://pandas.pydata.org/`

6 Überschrift auf Ebene 0 (chapter)

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

6.1 Überschrift auf Ebene 1 (section)

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

6.1.1 Überschrift auf Ebene 2 (subsection)

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Überschrift auf Ebene 3 (subsubsection)

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung,

wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Überschrift auf Ebene 4 (paragraph) Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

6.2 Listen

6.2.1 Beispiel einer Liste (itemize)

- Erster Listenpunkt, Stufe 1
- Zweiter Listenpunkt, Stufe 1
- Dritter Listenpunkt, Stufe 1
- Vierter Listenpunkt, Stufe 1
- Fünfter Listenpunkt, Stufe 1

Beispiel einer Liste (4*itemize)

- Erster Listenpunkt, Stufe 1
  - Erster Listenpunkt, Stufe 2
    - Erster Listenpunkt, Stufe 3
      - Erster Listenpunkt, Stufe 4
      - Zweiter Listenpunkt, Stufe 4
    - Zweiter Listenpunkt, Stufe 3
  - Zweiter Listenpunkt, Stufe 2

- Zweiter Listenpunkt, Stufe 1

6.2.2 Beispiel einer Liste (enumerate)

1. Erster Listenpunkt, Stufe 1
2. Zweiter Listenpunkt, Stufe 1
3. Dritter Listenpunkt, Stufe 1
4. Vierter Listenpunkt, Stufe 1
5. Fünfter Listenpunkt, Stufe 1

Beispiel einer Liste (4*enumerate)

1. Erster Listenpunkt, Stufe 1
    a) Erster Listenpunkt, Stufe 2
        i. Erster Listenpunkt, Stufe 3
            A. Erster Listenpunkt, Stufe 4
            B. Zweiter Listenpunkt, Stufe 4
        ii. Zweiter Listenpunkt, Stufe 3
    b) Zweiter Listenpunkt, Stufe 2
2. Zweiter Listenpunkt, Stufe 1

6.2.3 Beispiel einer Liste (description)

Erster Listenpunkt, Stufe 1

Zweiter Listenpunkt, Stufe 1

Dritter Listenpunkt, Stufe 1

Vierter Listenpunkt, Stufe 1

Fünfter Listenpunkt, Stufe 1

Beispiel einer Liste (4*description)

Erster Listenpunkt, Stufe 1

Erster Listenpunkt, Stufe 2

Erster Listenpunkt, Stufe 3

Erster Listenpunkt, Stufe 4

Zweiter Listenpunkt, Stufe 4

Zweiter Listenpunkt, Stufe 3

Zweiter Listenpunkt, Stufe 2

Zweiter Listenpunkt, Stufe 1

7 Zitieren

Der Zitierstil ist so angepasst, dass er den Zitierrichtlinien des Studiengangs Wirtschaftsinformatik der DHBW Stuttgart entspricht.

7.1 Zitate in den Text einfügen

In LaTeX wird mit den Befehlen `\footcite` oder `\cite` eine Referenz im Text eingefügt. Meist wird `\cite` nur *innerhalb* einer Fußnote benutzt. Damit ein vorangestelltes „Vgl.“ in der Fußnote erscheint, können Sie wie folgt zitieren:

```
\footcite[Vgl.][S. 3]{Autor}
\footcite[Vgl.][]{Autor}
```

Das erste optionale Argument von `\footcite` wird dem Zitat vorangestellt, das zweite ist die Seitenzahl. Den selben Effekt hätte

```
\footnote{Vgl. \cite[S. 3]{Autor}}
\footnote{Vgl. \cite{Autor}}
```

Hinweis: Falls „Vgl.“, aber keine Seitenzahl angeben werden soll, muss das zweite Argument vorhanden (jedoch leer) sein, ansonsten wird „Vgl.“ als Seitenzahl interpretiert. Falsch ist also:

```
\footcite[Vgl.]{Autor}  % so nicht!
```

7.1.1 Beispiele

Nachfolgend ein paar Beispiele, um die korrekte Darstellung zu überprüfen:

- **Schlosser** ist ein Buch über LaTeX.
- Zur Vorlesung *Logik und Algebra* gibt es das gleichnamige Lehrbuch.[188]
- nochmal dasselbe Buch[189]
- ein weiteres Buch desselben Autors[190]
- Der Konferenzbeitrag **Ancuti** beschäftigt sich mit Bildverarbeitung.

[188] **Staab**
[189] **Staab**
[190] **BuschlingerStaab**

- Cloud Computing wird in einer Diplomarbeit erklärt.[191]
- Preiß[192] gibt eine Einführung in Datenbanken.
- Eine Erläuterung, was „Intangibles“ sind, findet sich bei Stoi[193].
- weitere Ausführung in derselben Quelle[194]
- Laut Wikipedia[195] ist Wirtschaftsinformatik ein interessantes Studienfach.
- ITIL-Prozesse kann man tatsächlich auch mit LaTeX dokumentieren.[196]
- Open-Source und Cloud-Computing in einem Buchbeitrag[197]
- Buch mit zwei Autoren[198]
- Buch mit drei Autoren[199]
- Buch ohne Autor[200]
- Buch ohne Autor und ohne Jahr[201]
- und noch ein anderes Buch ohne Autor und ohne Jahr[202]
- Buch ohne Autor, aber dafür mit Herausgeber[203]
- manche Bachelorarbeit baut auf einer vorhergehenden Projektarbeit[204] auf
- das Handbuch zu BibLaTeX[205] und eines zu Windows 8[206]
- zwei Beiträge zu Büchern[207,208] und zu einem Konferenzband[209]
- eine Online-Quelle[210]
- eine plagiierte Dissertation,[211] nicht zur Nachahmung empfohlen

[191] **Boettger:Diplomarbeit**
[192] **Preiss**
[193] **Stoi**
[194] **Stoi**
[195] **wiki:Wirtschaftsinformatik**
[196] **Carvalho:PJ:2012-1**
[197] **Wind**
[198] **MitZweiAutoren**
[199] **MitDreiAutoren**
[200] **OhneAutoren**
[201] **OhneAutorenOhneJahr**
[202] **OhneAutorenOhneJahr2**
[203] **keinAutorAberHerausgeber**
[204] **mayer:PA1**
[205] **biblatex:manual**
[206] **Win8**
[207] **Trautwein:Nokia**
[208] **Mann**
[209] **Trautwein:Erfolgsfaktoren**
[210] **SAP:HANA**
[211] **GuttenPlag**

- zum Testen, ob Umlaute und Sonderzeichen korrekt wiedergegeben werden[212]

7.1.2 Spezialfälle

- *Zwei Quellen am Satzende* werden durch Komma getrennt.[213,214] Hier muss `${}^{,}$` eingeschoben werden.
- *Eindeutigkeit:* Normalerweise wird kein Vorname des Autors angegeben. Falls es allerdings zur Eindeutigkeit[215] (bei gleicher Jahreszahl) erforderlich ist, wird der Vorname abgekürzt bzw. nötigenfalls sogar ganz ausgeschrieben mit angegeben.[216]

  Welch ein Glück, dass Sie sich darum dank LaTeX gar nicht kümmern müssen (arme Word™-User ;-).
- Die Verwendung von *Sekundärliteratur*[217] wird weiter in Abschnitt 7.3 erläutert.

7.2 Eintragstypen für die Literatur-Datenbank

Die verwendete Literatur pflegen Sie in einer Literatur-Datenbank im Bibtex-Format. Dabei handelt es sich um eine Textdatei, wobei für jede Quelle mittels Name-Value-Pairs die relevanten Attribute (Autor, Titel etc.) hinterlegt sind. Die Datei wird üblicherweise nicht im Texteditor, sondern in einem spezialisierten Programm wie JabRef bearbeitet.

Sofern in der Literatur-Datenbank der Typ eines Eintrags (Entry Type) korrekt festgelegt ist, wird er im Literaturverzeichnis automatisch richtig dargestellt. Mit folgenden Typen sollten Sie i.d.R. auskommen:

article Artikel in einer Fachzeitschrift, auch E-Journal (Zeitschrift in elektronischer Form)[218]

book Buch, auch E-Book

inbook Kapitel in einem Buch, zu dem mehrere Autoren beigetragen haben

inproceedings Beitrag zu einer Fachtagung/Konferenz

manual Handbuch

misc anderweitig nicht zuordenbarer Typ

[212] **Umlauttest**
[213] **Staab**
[214] **mayerLukas:PA1**
[215] **trautwein2011unternehmensplanspiele** vs. **hitzler2011optimierung**
[216] Vgl. **mayer:PA1** und **mayerLukas:PA1**
[217] **Primaerquelle**, zitiert nach **Sekundaerquelle**
[218] Bei E-Journals/E-Books werden beim Zitieren anstelle der (u.U. nicht eindeutigen, da von der Schriftgröße abhängigen) Seitenzahl Abschnitt und Absatz näher bezeichnet, also: **Staab**.

phdthesis Dissertation

thesis Bachelor-/Master-/Diplomarbeit (Art wird im Attribut „type“ festgelegt)

online Internet- oder Intranet-Quelle[219]

report technischer Bericht, Forschungsbericht oder White Paper; diesen Typ können Sie auch verwenden, um eine Projektarbeit zu zitieren (Art wird im Attribut „type“ festgelegt)

Eine Übersicht über die notwendigen Attribute jedes Eintragstyps gibt die folgende Tabelle, wobei ein Schrägstrich als „oder“ zu verstehen ist.[220] Zudem sind die wichtigsten optionalen Attribute aufgeführt.

Eintragstyp	**notwendige Attribute**	**optionale Attribute (Auswahl)**
article	author, title, journal, year/date	volume, number, pages, month, note
book	author, title, year/date	publisher, edition, editor, volume/number, series, isbn, url
inbook	author, title, booktitle, year/date	bookauthor, editor, volume/number, series, isbn, url
inproceedings	author, title, booktitle, year/date	organization/publisher, editor, volume/number, series, isbn, url
manual	author/editor, title, year/date	organization/publisher, address, edition, month, note, url, urldate
misc	author/editor, title, year/date	howpublished, organization, month, note
phdthesis	author, title, institution, year/date	address, month, note
thesis	author, title, institution, type, year/date	address, month, note
online[219]	author/editor, title, year[221]/date, url	urldate
report	author, title, institution, type, year/date	number, version, url, urldate

7.3 Zitieren von Sekundärliteratur

Gelegentlich lässt es sich nicht vermeiden, aus der Sekundärliteratur zu zitieren. Dies leistet der folgende Befehl.

[219] Man beachte, dass der Eintragstyp „online“ in JabRef nur im „biblatex-Modus“ (Menü: Datei – Neue biblatex Bibliothek) auswählbar ist.

[220] Auszugsweise entnommen aus **biblatex:manual**.

[221] Sofern kein Jahr bekannt ist, sollte das Attribut nicht leer gelassen werden (sonst wird die aktuelle Jahreszahl automatisch eingefügt), sondern der Eintrag „o.J.“ gewählt werden.

```
\footcitePrimaerSekundaer{Primaerquelle}{Seite}{Sekundaerquelle}{Seite}
```

Die erste Seitenangabe bezieht sich auf die Primär-, die zweite auf die Sekundärquelle. Die Seitenangaben sind optional, sie können auch leer bleiben.[222] Es ist aber zu beachten, dass der Befehl `\footcitePrimaerSekundaer` vier Argumente hat.

Ins Literaturverzeichnis soll nur die Sekundärquelle aufgenommen werden. Dies wird dadurch erreicht, dass in der Literatur-Datenbank bei der Primärquelle im Attribut „keyword“ der Wert „ausblenden“ eintragen wird.

[222] **Primaerquelle**, zitiert nach **Sekundaerquelle**

8 Beispiele für Abbildungen und Tabellen

Hier finden Sie Beispiele für Abbildungen, Tabellen, Formelsatz und Source Code.

8.1 Abbildungen

In diesem Abschnitt gibt die Abbildungen 1 und 2, die beide das Logo der DHBW zeigen.

Abb. 1: DHBW-Logo 2cm hoch.[223]

Spezialfall: Sofern *innerhalb* der Bezeichnung einer Abbildung eine Fußnote angegeben oder eine Quelle referenziert werden soll, geschieht dies nicht per **footnote** oder **footcite**. Vielmehr sind die Befehle **footnotemark** und **footnotetext** zu verwenden und außerdem das optionale Argument für **caption** anzugeben (vgl. Source Code).

Abb. 2: DHBW-Logo 2cm breit. (Quelle: DHBW[224])

8.2 Tabellen

In diesem Abschnitt gibt es zwei Beispiel-Tabellen, nämlich auf Seite 43 und auf Seite 44.

links	Mitte	rechts
Muster	Muster	Muster

Tab. 1: Kleine Beispiel-Tabelle.

8.3 Etwas Mathematik

Eine abgesetzte Formel:

$$\int_a^b x^2 \,\mathrm{d}x = \frac{1}{3}(b^3 - a^3)$$

[223] Mit Änderungen entnommen aus: **OhneAutorenOhneJahr**
[224] `www.dhbw.de`

Spalte 1	Spalte 2	Spalte 3	Spalte 4	Spalte 5	Spalte 6
a	b	c	d	e	f
Test	Test, Test	Test, Test, Test			
1	2	3	4	5	6

Tab. 2: Größere Beispiel-Tabelle.

Es ist $a^2 + b^2 = c^2$ eine Formel im Text.

8.4 Source Code

Source Code-Blöcke können auf folgende Arten eingefügt werden:

Direkt im LaTeX-Source Code:

```
1 if(1 > 0) {
2   System.out.println("OK");
3 } else {
4   System.out.println("merkwuerdig");
5 }
```

oder eingefügt aus einer externen Datei.

```
1 public class HelloWorld {
2   public static void main(String[] args) {
3     if(args.length == 0) {
4       System.out.println("Hallo Sie!");
5     } else {
6       System.out.println("Hallo " + args[0] + "!");
7     }
8   }
9 }
```

Anhang

Anhangverzeichnis

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. 3: Mal wieder das DHBW-Logo.

Anhang 1/2: Etwas Source Code

```
1 public class HelloWorld {
2   public static void main(String[] args) {
3     if(args.length == 0) {
4       System.out.println("Hallo Sie!");
5     } else {
6       System.out.println("Hallo " + args[0] + "!");
7     }
8   }
9 }
```

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.[225] 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`

[225] Diese basieren auf Vorschlägen von Yannik Ehlert – vielen Dank dafür!

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.[226] 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[227] 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)
  - 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.[228]

Korrekturen sind wieder mit „(Fix)“ gekennzeichnet.

[226] `http://miktex.org/portable`
[227] `https://sourceforge.net/projects/biblatex-biber/files/biblatex-biber/current/binaries/Windows/`
[228] 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`.[229]
- `_dhbw_biblatex-config.tex`
  - (Fix) Es wurde das Problem behoben, dass im Literaturverzeichnis bei bestimmten Eintragstypen der Titel in Anführungszeichen steht.[230]

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.[231]

- `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.

[229] vgl. `https://github.com/semprag/biblatex-sp-unified/issues/23`
[230] Danke an Florian Eichin für seinen Hinweis.
[231] 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.

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)