UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering

Featurebasierte Fehlererkennung mittels Methoden des Machine Learnings

Masterarbeit

zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Informatik

vorgelegt von

Stefan Hermann Strüder

Erstgutachter:	Prof. Dr. Jan Jürjens Institut für Softwaretechnik
Zweitgutachter:	Dr. Daniel Strüber Chalmers University of Technology - Göteborg, Schweden (bis 02.2020) Radboud-Universität - Nijmegen, Niederlande

Koblenz, im Februar 2020

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Kurzfassung

Dies ist die Kurzfassung in Deutsch.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Abstract

This is the abstract in English.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Anmerkung

Diese Masterarbeit entstand in Teilen in Zusammenarbeit mit der Forschungsgruppe der Division of Software Engineering unter der Leitung von Thorsten Berger am Department of Computer Science and Engeneering der Chalmers Universität of Technology in Göteborg, Schweden.

Mein besonderer Dank gilt Thorsten Berger für die Ermöglichung und Finanzierung dieser Zusammenarbeit. Ebenfalls gilt mein Dank dem gesamten Team der Forschungsgruppe für die Unterstützung bei Problemen und Fragen zu meiner Arbeit. Ein weiterer Dank gilt Daniel Strüber für seine Initiative zur Ermöglichung der Zusammenarbeit.

Comment

This master thesis was partly written in cooperation with the research group of the Division of Software Engineering headed by Thorsten Berger at the Department of Computer Science and Engineering of Chalmers University of Technology in Gothenburg, Sweden.

My special thanks goes to Thorsten Berger for facilitating and financing this cooperation. I would also like to thank the entire team of the research group for their support in case of problems and questions concerning my work. A further thank you goes to Daniel Strüber for his initiative to make this cooperation possible.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung und Motivation

Ausblick: Dieses Kapitel dient zur allgemeinen Einführung in diese Masterarbeit. Dazu werden neben einer Einleitung und Motivation in das zugrundeliegende Thema, die grundlegenden Strukturen der Arbeit erläutert. Dazu gehören die Forschungsziele und Forschungsfragen, das verwendete Forschungsdesign, eine Übersicht der angedachten und tatsächlichen Zeitplanung sowie eine Erläuterung des Aufbaus der weiterführenden Teile dieser Arbeit.

Softwarefehler stellen einen erheblichen Auslöser für finanzielle Schäden und Rufschädigungen von Unternehmen dar. Solche Fehler reichen von kleineren „Bugs“ bis hin zu schwerwiegenden Sicherheitslücken. Aus diesem Grund herrscht ein großes Interesse daran, einen Entwickler zu warnen, wenn er aktualisierten Softwarecode veröffentlicht, der möglicherweise einen Fehler beinhaltet.

Zu diesem Zweck haben Forscher und Softwareentwickler im vergangenen Jahrzehnt verschiedene Techniken zur Fehlererkennung und Fehlervorhersage entwickelt, die zu einem Großteil auf Methoden und Techniken des *Machine Learnings* basieren. Diese verwenden in der Regel historische Daten von fehlerhaften und fehlerfreien Änderungen an Softwaresystemen in Kombination mit einer sorgfältig zusammengestellten Menge von *Attributen* (in der Regel Features genannt [1]), um einen gegebenen Klassifikator anzulernen beziehungsweise zu trainieren. Dieser dient dann dazu, ein akkurate Vorhersage zu treffen, ob eine neu erfolgte Änderung an einer Software fehlerbehaftet oder frei von Fehlern ist.

Die Auswahl an Lernverfahren für Klassifikatoren ist groß. Studien zeigen, aus diesem Pool von Verfahren sowohl Entscheidungsbaum-basierte (zum Beispiel J48, CART oder Random Forest) als auch bayessche Verfahren die meistgenutzten sind [4]. Alternative Lernmethoden sind Regression, k-Nearest-Neighbor oder künstliche neuronale Netze [2]. Anzumerken ist allerdings, dass es keinen Konsens über die beste verfügbare Lernverfahren gibt, da jedes Verfahren unterschiedliche Stärken und Schwächen für bestimmte Anwendungsfälle aufweist.

Das Ziel dieser Arbeit ist die Entwicklung einer solchen Vorhersagetechnik für Softwarefehler basierend auf Software-Features. Diese beschreiben Inkremente der Funktionalität eines Softwaresystems. Die auf diese Weise entwickelten Softwaresysteme heißen Software-Produktlinien und bestehen aus einer Menge von ähnlichen Softwareprodukten. Sie zeichnen sich dadurch aus, dass sie eine gemeinsame Menge von Features sowie eine gemeinsame Codebasis besitzen

[1] Um einem missverständlichen und doppeldeutigen Gebrauch des Feature-Begriffes vorzubeugen, wird für die hier verwendete Beschreibung der Charakteristika von Daten auch im weiteren Verlauf dieser Ausarbeitung der Begriff „Attribute“ verwendet.

[5]. Durch das Vorhandensein verschiedener Features entlang der Softwareprodukte, kann eine breite Variabilität innerhalb einer Produktlinie erreicht werden.

ÜBERARBEITEN!

Die nachfolgende Abbildung 1.1 zeigt den zentralen Prozess der Entwicklung einer Produktlinie. Aufgeteilt wird dieser in das Domain Engineering und das Application Engineering. Im Rahmen des Domain Engineerings wird ein sogenanntes Variabilitätsmodell (Variability Model) erzeugt, welches durch die Kombination der wählbaren Features beschrieben wird [1]. Gängige Implementationstechniken für Features reichen von einfachen Lösungen durch Annotationen basierend auf Laufzeitparametern oder Präprozessor-Anweisungen bis hin zu verfeinerten Lösungen basierend auf erweiterten Programmiermethoden, wie zum Beispiel Aspektorientierung. In Teilen dieser Implementierungstechniken wird jedes Feature wird als wiederverwendbares Domain Artifact modelliert und gekapselt, welches im Prozess des Application Engineerings in Form einer Konfiguration zusammen mit weiteren Features, im Hinblick auf die gewünschte Funktionalität der Software, ausgewählt werden kann. Ein Software Generator erzeugt dann die gewünschten Software Produkte basierend auf den bereits zuvor genannten Implementationstechniken für Features.

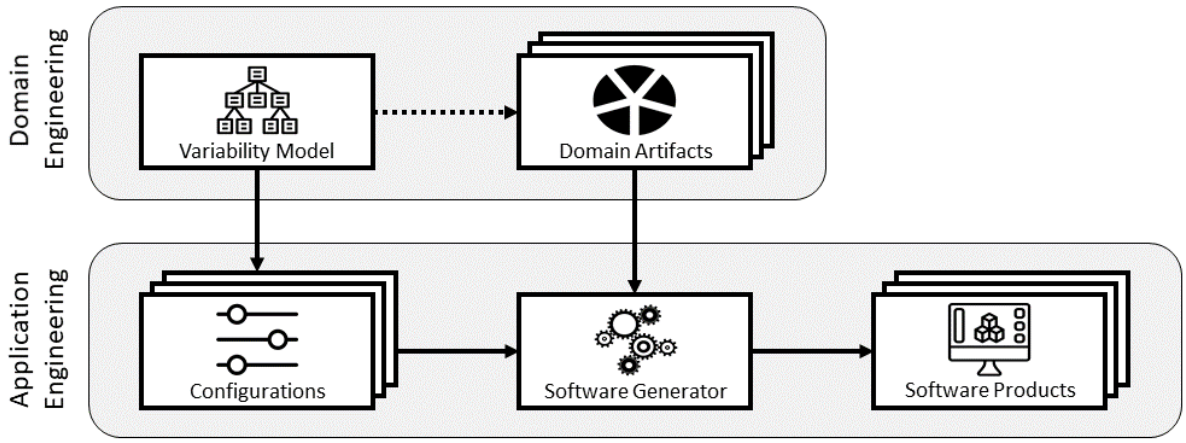

Abbildung 1.1: Generierung von Software-Produktlinien nach [5]

Das Ziel dieser Arbeit ist die Entwicklung einer auf Machine Learning gestützten Vorhersagetechnik für defekte Software-Features. Dieser bisher in wissenschaftlichen Papern nur einmal betrachtete Ansatz ist aufgrund mehrerer Gründe chancenreich:

1. Wenn ein bestimmtes Feature in der Vergangenheit mehr oder weniger fehleranfällig war, so ist eine Änderung, die das Feature aktualisiert, wahrscheinlich ebenfalls mehr oder weniger fehleranfällig.

2. Features, die mehr oder weniger fehleranfällig scheinen, könnten besondere Eigenschaften haben, die im Rahmen der Fehlervorhersage verwendet werden können.

3. Code, der viel Feature-spezifischen Code enthält (insbesondere die sogenannten Feature-Interaktionen), ist möglicherweise fehleranfälliger als sonstiger Code.

Das zuvor genannte Ziel der Arbeit setzt sich aus mehreren Teilzielen zusammen. Dazu zählen die Erstellung eines Datensets zum Trainieren von Machine-Learning-Klassifikatoren sowie

das Anlernen einer repräsentativen Auswahl an Klassifikatoren mit anschließender vergleichender Evaluation dieser. Ein genauer Überblick über die Forschungsziele befindet sich im nächsten Unterkapitel.

Sollte sich im Rahmen der Evaluation einer dieser Klassifikatoren als besonders effektiv erweisen, so würde diese Arbeit den Stand der Technik hinsichtlich der Fehlererkennung in Features vorantreiben und Organisationen erlauben, bessere Einblicke in die Fehleranfälligkeit von Änderungen in ihrer Codebasis zu erhalten.

1.1 Forschungsziele und Forschungsfragen

ÜBERARBEITEN!

Wie bereits in der Einleitung beschrieben, ist das übergeordnete Ziel dieser Arbeit die Entwicklung einer Vorhersagetechnik für Fehler in featurebasierter Software unter Zuhilfenahme von Methoden des Machine Learnings. Dazu ist vorgesehen, das Augenmerk auf Commits von Versionierungssystemen, wie beispielsweise Subversion oder Git, zu richten. Ein Commit bezeichnet dabei die zur Verfügungstellung einer aktualisierten Version einer Software. Als Datenbasis für das Trainieren der Klassifikatoren dienen dann fehlerhafte und fehlerfreie Commits von featurebasierter Software. Dies ermöglicht es, ausstehende defekte Commits vorherzusagen und das Risiko der Konsequenzen von Softwarefehlern zu senken.

Der Prozess der Entwicklung der Vorhersagetechnik ist in drei zu erreichende Forschungsziele eingeteilt. Jedem Forschungsziel werden Forschungsfragen zugeordnet, deren Aufklärung einen zusätzlichen Teil zur Erfüllung der Ziele beiträgt. Im Folgenden werden die Forschungsziele (RO – „research objective“) mit ihren zugehörigen Forschungsfragen (RQ – „research question“) vorgestellt.

RO1: Erstellung eines Datensets zum Trainieren von relevanten Machine-Learning-Klassifikatoren

RQ1a: Welche Daten kommen für die Erstellung des Datensets in Frage?
RQ1b: Wie weit müssen die Daten vorverarbeitet werden, um sie für das Training nutzbar zu machen?

RO2: Identifikation und Training einer Auswahl von relevanten Machine Learning Klassifikatoren basierend auf dem Datenset

RQ2: Welche Machine Learning Klassifikatoren kommen für die gegebene Aufgabe in Frage?

RO3: Evaluierung und Gegenüberstellung der Klassifikatoren sowie Vergleich zu modernen Vorhersagetechniken, die keine Features nutzen

RQ3a: Welche miteinander vergleichbaren Merkmale besitzen die Klassifikatoren?
RQ3b: Welche Metriken können für den Vergleich verwendet werden?
RQ3c: Welche Vor- und Nachteile besitzt ein Klassifikator?
RQ3d: Wie lassen sich die Klassifikatoren mit weiteren Vorhersagetechniken, die keine Features nutzen, vergleichen?

Zusätzlich zu den drei genannten Forschungszielen umfasst die Bearbeitung der Masterarbeit eine Vor- und Nachbereitung, sodass sich insgesamt fünf Arbeitsphasen ergeben. Diese werden in den weiteren Unterkapiteln näher erläutert. Als finale Vorhersagetechnik wird jener Klassifikator verwendet, der sich im Rahmen der Gegenüberstellung im Verlauf der Evaluation als am effektivsten erweist.

1.2 Forschungsdesign

Die für diese Arbeit gewählte Methodik basiert auf dem Prozessmodell Cross-Industry Standard Process for Data Mining, kurz CRISP-DM, nach [3]. Es wird als Vorlage für die Arbeitsphasen zur Erreichung der Forschungsziele dieser Arbeit verwendet. Da der überwiegende praktische Teil dieser Arbeit auf Programmierung im Bereich des Machine Learning konzentriert, bildet das CRISP-DM Prozessmodell ein passendes vordefiniertes Vorgehen. Die nachfolgende Abbildung 1.2 zeigt eine grafische Aufarbeitung des Prozessmodells mit seinen sechs zugehörigen Phasen sowie den Verbindungen zwischen diesen.

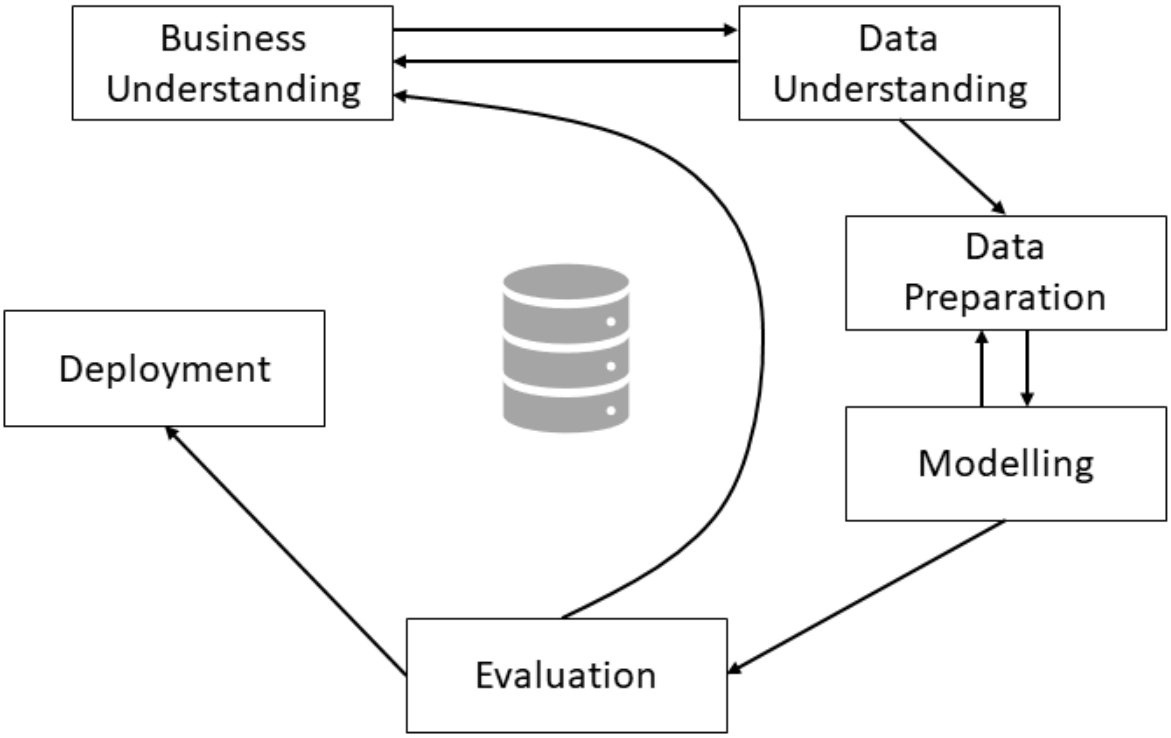

Abbildung 1.2: CRISP-DM Prozessmodells nach [3]

Das CRISP-DM Prozessmodell würde ursprünglich, wie der Name bereits andeutet, für die Erarbeitung von Data Mining Projekten entwickelt, eignet sich jedoch auch zur Verwendung im Rahmen eines Machine Learning Projektes, da sich die in beiden Bereichen verwendeten Methoden und Prozesse zu einem erheblichen Teil überlagern. Ein Überblick über die sechs Phasen des Prozessmodells ist in Abbildung 1.3 dargestellt. Zusätzlich umfasst diese Abbildung die Zuordnung der Arbeitsphasen, die im vorherigen Unterkapitel definiert wurden. Eine Erläuterung der Phasen des Prozessmodells erfolgt im Anschluss der Abbildung. Einen genauen Überblick über den konkreten Umfang der Arbeitsphasen, aufgeteilt in jeweilige Unterziele und zu erfüllende Aufgaben, bietet das im Anschluss folgende Unterkapitel.

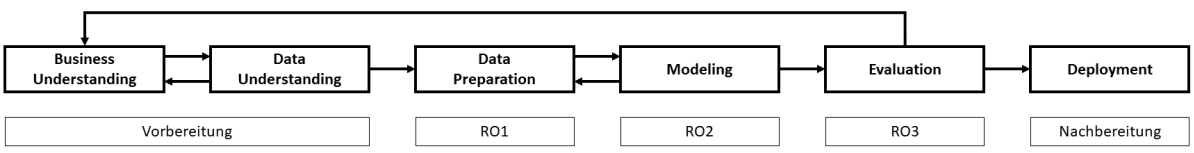

Abbildung 1.3: Phasen des CRISP-DM Prozessmodells nach [3] mit Zuordnung der Arbeitsphasen

Die ersten beiden Phasen *Business Understanding* und *Data Understanding* widmen sich der Vorbereitung der Arbeit. Die initiale Phase umfasst dabei die allgemeine Einarbeitung in das zugrundeliegende Thema und der Formulierung der Forschungsziele. Anzumerken ist, dass diese Phase bereits vor der sechsmonatigen Bearbeitungszeit der Arbeit beginnt und somit schon das Verfassen dieses Proposals als Teilaufgabe dieser Phase gezählt werden kann, da es auch eine grobe Einarbeitung in das Thema erfordert.

Die darauffolgende Phase *Data Understanding* dient der Suche und Einsicht von für den weiteren Verlauf der Phasen relevanten Daten und, falls vorhanden, vorgefertigten Datensets. Da Commits als Datenbasis zur Erlernung der Klassifikatoren betrachtet werden, wird der überwiegende Teil der Suche nach Daten auf dem Onlinedienst GitHub stattfinden, welchem das Versionierungssystemen Git zugrunde liegt. Für die weiteren Phasen ist es von besonderer Bedeutung, den Aufbau der Daten sorgfältig zu untersuchen.

Die dritte Phase *Data Preparation* kümmert sich um die Erstellung eines endgültigen Datensets und den dort hinführenden Prozessen. Diese Phase ist deckungsgleich mit den Anforderungen des ersten Forschungsziels.

Zur Anwendung kommt das im vorherigen Schritt erstellte Datenset in der Phase *Modeling*. In dieser werden die Data-Mining-Algorithmen und -Techniken gemäß den zugrundeliegenden Anforderungen auf das Datenset angewendet. Adaptiert an das Lernen der Machine Learning Klassifikatoren spiegelt dies die Arbeitsphase zur Erfüllung des zweiten Forschungsziels dar.

Die fünfte Phase umfasst die *Evaluation* der Resultate des zuvor erfolgten Schrittes und deckt somit die Erfüllung des dritten Forschungsziels ab.

Die Nachbereitung der Arbeit wird durch die Phase *Deployment* abgedeckt. Diese umfasst die Erstellung der finalen Ausarbeitung sowie der Abschlusspräsentation und der anschließenden Vorführung dieser im Rahmen des Kolloquiums.

Es ist zu erkennen, dass die Beschreibung der Arbeitsphasen weitestgehend auf einem theoretischen Level verfasst wurde. Es wird anhand der fünf CRISP-DM-Phasen gezeigt, was für den erfolgreichen Abschluss der Arbeit absolviert werden muss. Die Erörterung der Frage, wie die einzelnen zu erledigenden Aufgaben durchgeführt werden müssen, ist Teil der Vorbereitung der Arbeit. Im Rahmen der Phasen Business Understanding und Data Understanding wird nach einer eingehenden Recherche die genaue Arbeitsplanung festgelegt (siehe Unterziele der ersten Phase im folgenden Unterkapitel).

1.3 Zeitplanung

ANPASSEN AN TATSÄCHLICHEN ABLAUF

Im Nachfolgenden wird die vorläufige Ablaufplanung der Arbeit aufgezeigt. Die Ordnung erfolgt gemäß der Aufteilung in die fünf zuvor beschriebenen Arbeitsphasen. Die geschätzte Dauer der verschiedenen Unterziele wird jeweils in Tagen, Wochen oder Monaten angegeben. Zur Verfügung stehen insgesamt sechs Monate Bearbeitungszeit.

Phase 1: Vorbereitung

Unterziele	geplante Dauer	tatsächliche Dauer	
Strukturierte Literaturrecherche - Techniken der featurebasierten Softwareprogrammierung - Techniken zur Fehlererkennung in Software - Klassifikation mittels Machine Learning - Klassifikationsmethoden - Auswahl der Programmiersprache - Tool- und Libraryauswahl - Evaluationsmetriken	2 Wochen	TBD	Business Understanding
Recherche zur Bildung eines Datensets - Merkmale / Aufbau eines Datensets - Suche nach Datenquellen - Suche nach vorgefertigten Datensets - Prüfung der Daten / Datensets auf Eignung - Analyse des Aufbaus der Daten / der Datensets	1 Woche	TBD	Data Understanding
Total:	3 Wochen	TBD	

Phase 2: Forschungsziel 1 – Erstellung des Datensets (Data Preparation)

Unterziele	geplante Dauer	tatsächliche Dauer
finale Datenauswahl - Festlegung von Kriterien	1 Woche	TBD
Datenbereinigung - "Preprocessing"	1 Woche	TBD
finale Konstruktion des Datensets - Integration der Daten und des Feature-Aspekts - erneute abschließende Bereinigung sowie Formatierung - Teilung in Training-Set und Test-Set	1 Woche	TBD
Total:	3 Wochen	TBD

Phase 3: Forschungsziel 2 – Training der Machine Learning Klassifikatoren (Modeling)

Unterziele	geplante Dauer	tatsächliche Dauer
Auswahl geeigneter Klassifikatoren	1 Woche	TBD
Training der Klassifikatoren	3 Wochen	TBD
Total:	4 Wochen	TBD

Phase 4: Forschungsziel 3 – Evaluation und Vergleich der Machine Learning Klassifikatoren

Unterziele	geplante Dauer	tatsächliche Dauer
Evaluation der einzelnen Klassifikatoren - Festlegung der Bewertungsmetriken - Anwendung des Test-Sets - Berechnung der Bewertungsmetriken	2 Wochen	TBD
Vergleich der Klassifikatoren anhand der Metriken	2 Wochen	TBD
Vergleich mit weiteren Vorhersagetechniken, die nicht auf Features setzen	1 Woche	TBD
Total:	5 Wochen	TBD

Phase 5: Nachbereitung (Deployment

Unterziele	geplante Dauer	tatsächliche Dauer
Besprechung der vorangegangenen Arbeit mit Betreuer - Umsetzung möglicher Verbesserungsvorschläge	1 Woche	TBD
Erstellung der Ausarbeitung	7 Wochen	TBD
Erstellung der Abschlusspräsentation	1 Woche	TBD
Total:	9 Wochen	TBD

* Die Erstellung der Ausarbeitung ist ein laufender Prozess über den gesamten Verlauf der Bearbeitungszeit. Der hier erwähnte siebenwöchige Zeitraum dient unter Anderem zur Korrektur beziehungsweise Verbesserung hinsichtlich des Feedbacks des Betreuers und zur abschließenden Finalisierung.

Die nachfolgende Abbildung zeigt den zeitlichen Ablauf der Arbeit als Gantt-Chart inklusive konkreter Datumsangaben. Eine größere Version des Plans befindet sich im Anhang.

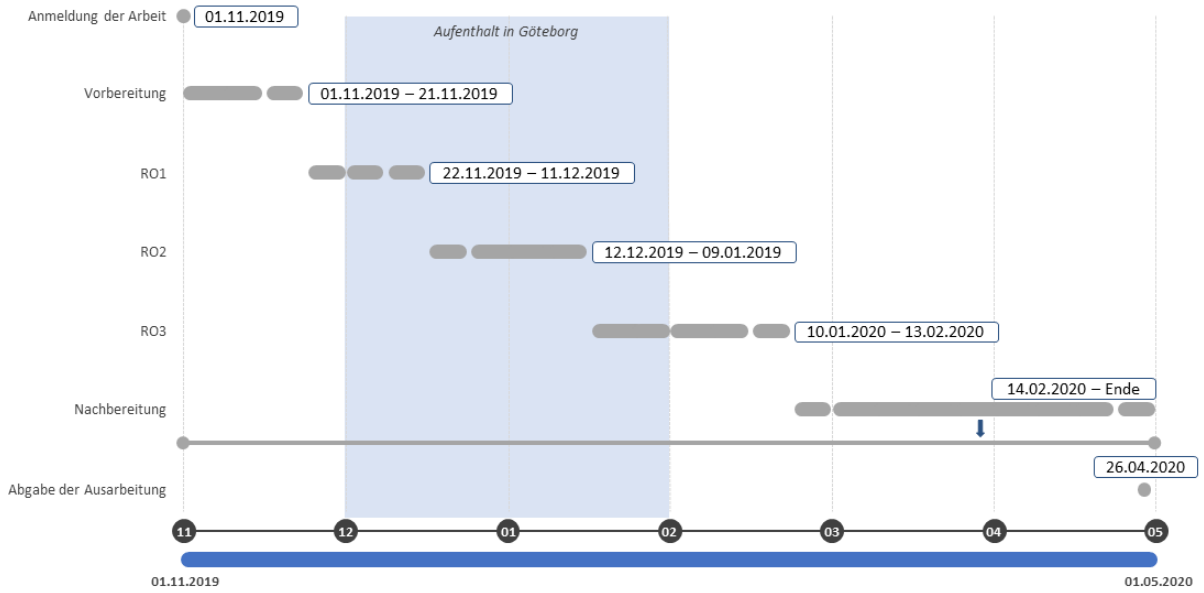

Abbildung 1.4: Zeitlicher Ablaufplan der Arbeit als Gantt-Chart

1.4 Aufbau der Arbeit

Diese Ausarbeitung ist in sechs Kapitel unterteilt. Das erste Kapitel, welches mit diesem Abschnitt abgeschlossen wird, diente zur Einführung in das Thema der Masterarbeit. Ebenso stellte es die theoretischen Rahmenbedingungen der Arbeit vor. Das zweite Kapitel „Hintergrund“ dient zur Vermittlung von Basiswissen zu den grundlegenden Themenkomplexen dieser Ausarbeitung. Dazu wird zunächst die featurebasierte Softwareentwicklung vorgestellt, ehe dann die Machine-Learning-Klassifikation sowie die darauf aufbauende Fehlervorhersage erläutert werden. Die zwei darauffolgenden Kapitel widmen sich der Auseinandersetzung des praktischen Teils dieser Masterarbeit in Form der Erstellung des featurebasierten Datensets sowie des Trainings der Machine-Learning-Klassifikatoren. Die Gegenüberstellung und Evaluation dieser Klassifikatoren erfolgt im fünften Kapitel inklusive eines Vergleiches zu nicht-featurebasierten Methoden zur Fehlererkennung. Eine abschließende Zusammenfassung sowie ein Ausblick auf weiterführende Projekte, die auf diese Masterarbeit aufbauen können, erfolgen im abschließenden sechsten Kapitel.

Zusätzlich wird die Ausarbeitung von zahlreichen Abbildungen zur verständlicheren Verdeutlichung von Zusammenhängen ergänzt.

Kapitel 2

Hintergrund

Ausblick: Zum besseren Verständnis der weiteren Verlaufs dieser Arbeit, dient dieses Kapitel zur Einführung in die zugrundeliegenden Themen. Dazu wird zunächst die featurebasierte Softwareentwicklung erläutert, ehe dann der Themenbereich des Machine Learnings vorgestellt wird. Dazu werden die Klassifikation und die Fehlervorhersage mittels Machine Learning erläutert. Unterstützt werden die Abschnitte von Grafiken zum besseren Verständnis der Zusammenhänge.

2.1 Featurebasierte Softwareentwicklung

2.2 Machine-Learning-Klassifikation

ÜBERARBEITEN!

Die Machine-Learning-Klassifikation unterliegt dem Teilgebiet des *überwachten Machine Learnings* (englisch: supervised Machine Learning). Die nachfolgende Abbildung 2.1 präsentiert den allgemeinen Prozess des überwachten Machine Learnings auf vereinfachter Weise anhand eines Beispiels. Anhand dieser werden die wichtigsten Informationen zum genannten Themengebiet erläutert.

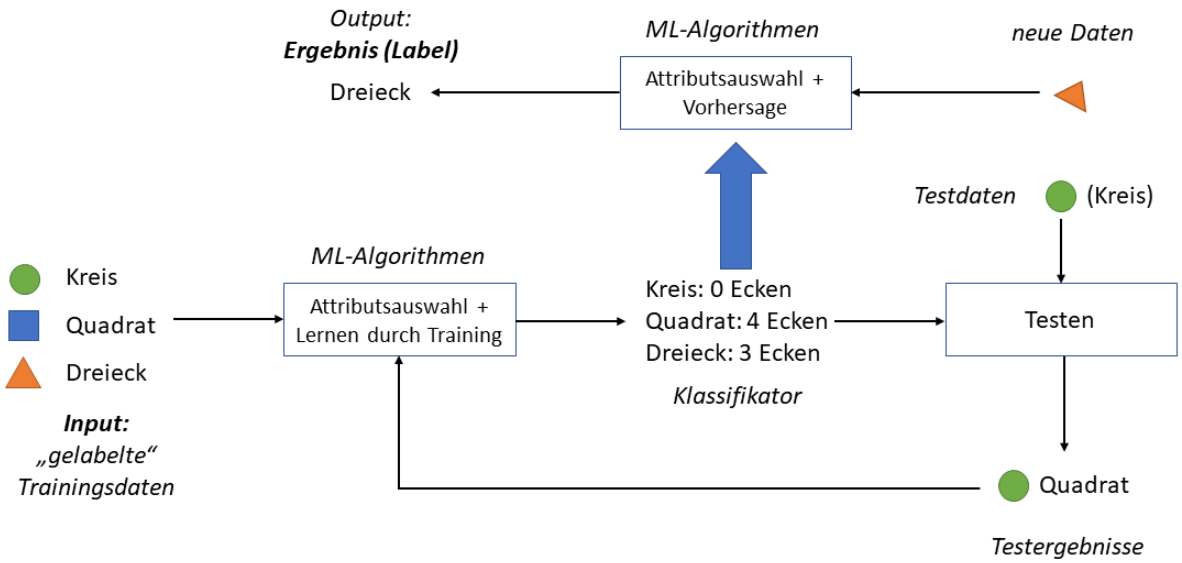

Abbildung 2.1: Allgemeiner Prozess des überwachten Machine Learnings dargestellt anhand eines Beispiels (vereinfacht)

Das in der Abbildung gezeigte Beispiel zeigt den Prozess der Entwicklung und Anwendung eines Klassifikators zur Erkennung von geometrischen Formen. Der Prozess beginnt mit der Erstellung eines Datensets, welches als Input für die Anlernung des Klassifikators dient.

ÜBERARBEITEN!

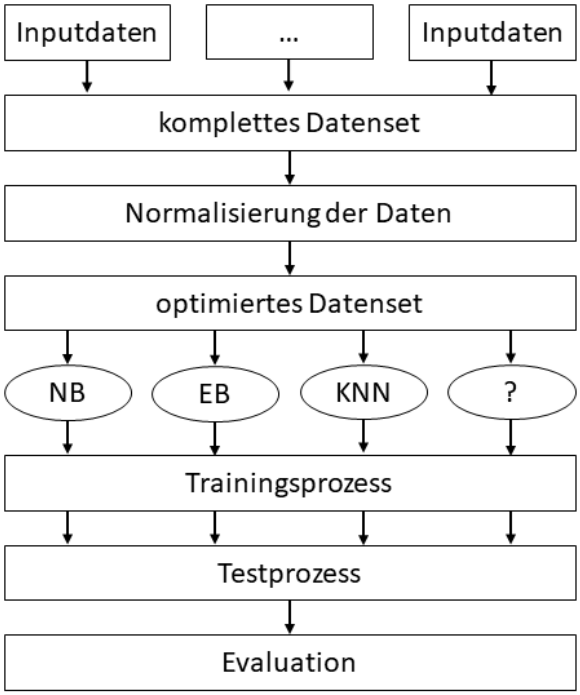

Abbildung 2.2: Angewendeter Prozess zur Durchführung der Klassfikation nach [**Ceylan2006**]

2.3 Fehlervorhersage mittels Machine Learning

Die nachfolgenden drei Abbildungen zeigen den von Queiroz et at. [**Queiroz2016**] angewandten Prozess zur Entwicklung und Anwendung eines featurebasierten Klassifikators im Rahmen des überwachten Machine Learnings. Die gezeigten Darstellungen orientieren sich sowohl gestalterisch als auch inhaltlich an den in Abbildung 2.1 gezeigten allgemeinen Prozess des überwachten Machine Learnings. Ferner dient dieser Prozess als grundlegender Prozess für diese Arbeit.

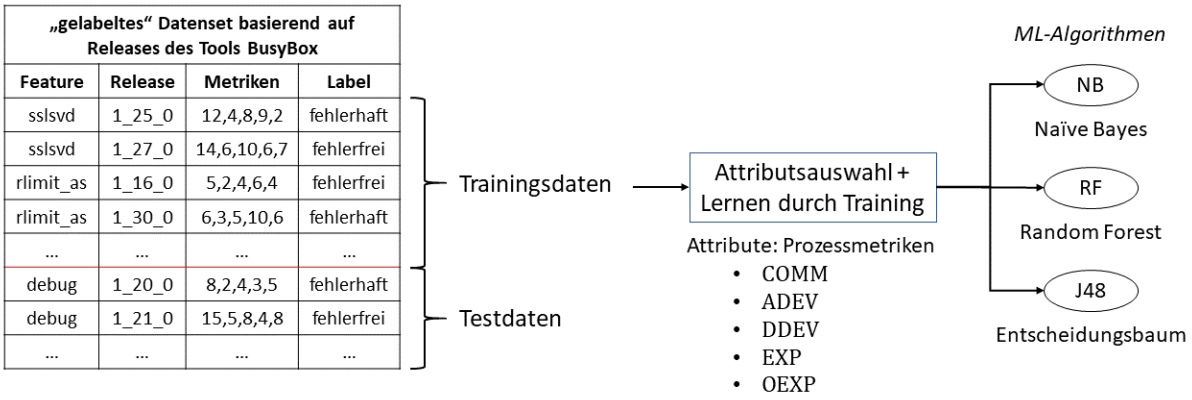

Abbildung 2.3: Teil 1: Featurebasierter Prozess des überwachten Machine Learnings nach [**Queiroz2016**]

Die Datenbasis des Datensets bilden Commits des UNIX-Toolkits BusyBox[1], dessen Quellcode frei verfügbar in einem Git-Repository[2] eingesehen und von dort geklont werden kann. Diese Commits wurden wiederum ihren entsprechenden Releases zugeordnet, welche auf der vergebenen Tag-Struktur des Repositories beruhen. Ferner wurden aus den Diffs der Commits die dort bearbeiteten Features extrahiert und anschließend zusammen mit den Release-Informationen in einer MySQL-Datenbank gespeichert. Zusätzlich enthält jeder Datenbankeintrag aggregierte Werte von fünf auf das Feature und den Release bezogenen Prozessmetriken (Erläuterung folgt), sowie das binäre Label, ob ein Feature in einem Release fehlerhaft oder fehlerfrei war. Ein Feature gilt in einem Release als fehlerhaft, sofern in einem Commit des darauffolgenden Releases ein fehlerbehebender Commit festgestellt werden konnte. Dies geschieht über die Analyse der Commit-Nachrichten. Sofern eine Commit-Nachricht die Begriffe "Bug", Ërror", "Failöder "Fixënthält, werden die Autoren des Papers den Commit als fehlerbehebend. Wie im Rahmen des überwachten Machine Learning üblich, wird das Datenset in Trainings- und Testdaten in einem Verhältnis von 75:25 geteilt.

Die Trainingsdaten werden dann den Klassifikatoren zur Anlernung zur Verfügung gestellt. Als Attribute dienen die fünf bereits erwähnten Prozessmetriken. Die nachfolgende Tabelle gibt einen Überblick über die Beschreibungen dieser.

[1] `https://busybox.net/`
[2] `https://git.busybox.net/busybox/`

Tabelle 2.1: Übersicht der verwendeten Prozessmetriken

Metrik	Beschreibung
COMM	Anzahl der Commits, die in einem Release dem betreffenden Feature / der betroffenen Datei gewidmet sind
ADEV	Anzahl der Entwickler, die das betreffende Feature / die betreffende Datei in einem Release bearbeitet haben
DDEV	kummulierte Anzahl der Entwickler, die das betreffende Feature / die betreffende Datei in einem Release bearbeitet haben
EXP	Geometrisches Mittel der Ërfahrungäller Entwickler, die am betreffenden Feature / an der betreffenden Datei in einem Release gearbeitet haben
OEXP	Ërfahrung"des Entwicklers, der am meisten zum betreffenden Feature / zur betreffenden Datei in einem Release beigetragen hat

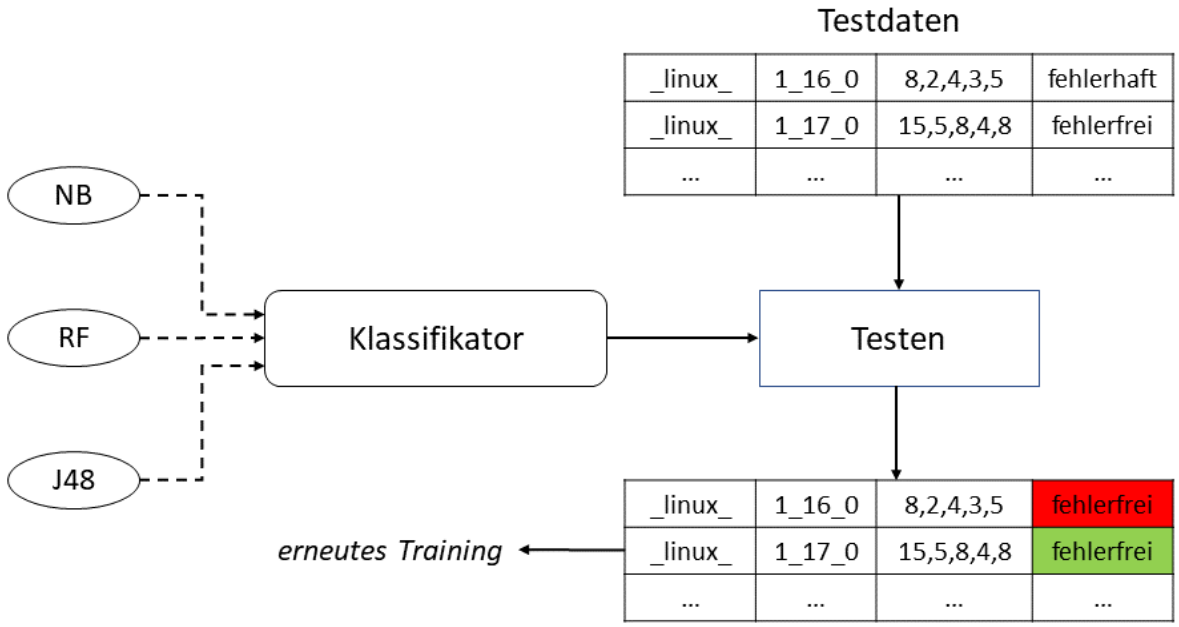

Abbildung 2.4: Teil 2: Featurebasierter Prozess des überwachten Machine Learnings nach **[Queiroz2016]**

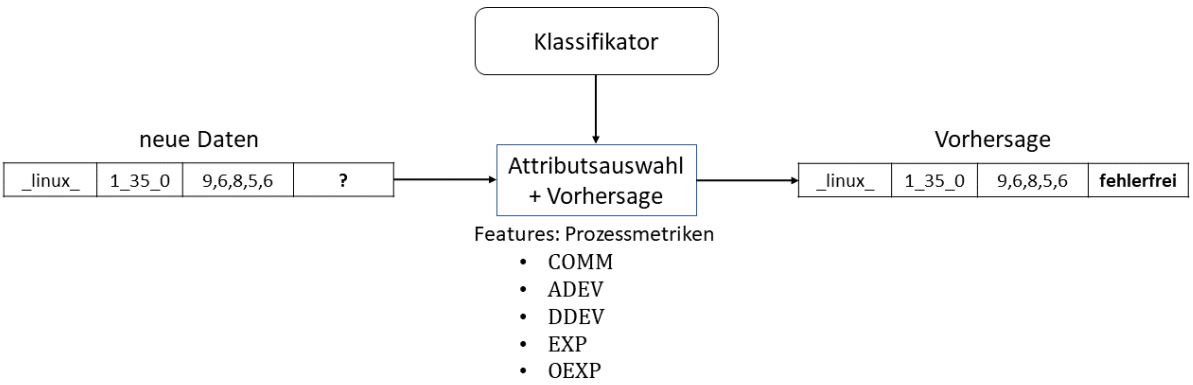

Abbildung 2.5: Teil 3: Featurebasierter Prozess des überwachten Machine Learnings nach **[Queiroz2016]**

Kapitel 3

Erstellung eines featurebasierten Datensets

Ausblick: Dieses Kapitel widmet sich der schrittweisen Erläuterung des Prozesses zur Erstellung des featurebasierten Datensets, welches zur Anlernung der Machine-Learning-Klassifikatoren dient. Dazu wird zunächst die Datenauswahl näher beleuchtet. Darauf folgt eine Darlegung der Konstruktion des Datensets sowie der Auswahl und Berechnung der Metriken, welche als Attribute (Features) im Rahmen der Anlernung der Klassifikatoren dienen.

3.1 Datenauswahl

Wie im vorangegangenen Kapitel bereits erwähnt wurde, bildet das Datenset die Grundlage für die Anlernung der Machine-Learning-Klassifikatoren und wird eigens für diese Arbeit auf Basis von Commits von 13 featurebasierten Software-Projekten erstellt. Die Auswahl der Software-Projekte erfolge anhand von vorheriger Verwendung in wissenschaftlicher Literatur [**Hunsen2015**, **Liebig2010**, **Queiroz2016**]. Die für diese Arbeit verwendeten Software-Projekte sind samt ihres Einsatzzweckes und ihrer Datenquellen in Tabelle XX aufgeführt.

Zum Erhalt der Commit-Daten der Software-Projekte wurde die Python-Library PyDriller[2] verwendet [**Spadini2018**]. Diese ermöglicht eine einfache Datenextraktion von Git-Repositories zum Erhalt von Commits, Commit-Nachrichten, Entwicklern, Diffs und mehr. Ein beispielhafter Sourcecode-Ausschnitt zur Konsolenausgabe von Metadaten eines Commits (Autor, Name der veränderten Dateien, Typ der Veränderung und jeweilige zyklomatische Komplexität der Dateien) ist in Listing 3.1 aufgeführt.

[1] Links zu den Websites der Softwareprojekte und deren Repositories können im Anhang eingesehen werden.
[2] https://github.com/ishepard/pydriller

Tabelle 3.1: Übersicht der verwendeten Software-Projekten[1]

	Zweck	Datenquelle		Zweck	Datenquelle
Blender	3D-Modellierungstool	GitHub-Mirror	**libxml2**	XML-Parser	GitLab-Repository
Busybox	UNIX-Toolkit	Git-Repository	**lighttpd**	Webserver	Git-Repository
Emacs	Texteditor	GitHub-Mirror	**MPSolve**	Polynomlöser	GitHub-Repository
GIMP	Bildbearbeitung	GitLab-Repository	**Parrot**	virtuelle Maschine	GitHub-Repository
Gnumeric	Tabellenkalkulation	GitLab-Repository	**Vim**	Texteditor	GitHub-Repository
gnuplot	Plotting-Tool	GitHub-Mirror	**xfig**	Grafikeditor	Sourceforge-Repository
Irssi	IRC-Client	GitHub-Repository			

Tabelle 3.2: Übersicht der Anzahl der Releases und Commits je Software-Projekt

	#Releases	#Commits		#Releases	#Commits
Blender	11	19119	**libxml2**	10	732
Busybox	14	4984	**lighttpd**	6	2597
Emacs	7	12805	**MPSolve**	8	668
GIMP	14	7240	**Parrot**	7	16245
Gnumeric	8	6025	**Vim**	7	9849
gnuplot	5	6619	**xfig**	7	18
Irssi	7	253			

```
for commit in RepositoryMining("link_to_repo").traverse_commits():
  for m in commit.modifications:
    print(
          "Author {}".format(commit.author.name),
          " modified {}".format(m.filename),
          " with a change type of {}".format(m.change_type.name),
          " and the complexity is {}".format(m.complexity)
    )
```

Listing 3.1: Beispielhafter PyDriller-Code zur Ausgabe von Metadaten von Commits

Als Input der Python-Scripte zum Erhalt der Commit-Daten dienten jeweils die URLs zu den Git-Repositories der Software-Projekte. Weiterhin wurden die Daten in Commits je Release aufgeteilt. Durchgeführt wurde dies durch die Angabe von Release-Tags, basierend auf der Tag-Struktur von Git-Repositories, im PyDriller-Code. Für jede veränderte Datei innerhalb eines Commits und eines Releases wurden die folgenden Metadaten mit Hilfe von PyDriller abgerufen:

- Commit-Hash (eindeutiger Bezeichner des zugehörigen Commits)
- Autor des zugehörigen Commits
- zugehörige Commit-Nachricht
- Name der veränderten Datei
- Lines-of-Code der veränderten Datei
- zyklomatische Komplexität der veränderten Datei
- Anzahl der hinzugefügten Zeilen zur Datei
- Anzahl der entfernten Zeilen von der Datei
- Art der Änderung (ADD, REM, MOD)[3]
- Diff der Veränderung

Die auf diese Weise erhaltenen Daten wurden nach dem Abruf in einer MySQL-Datenbank gespeichert. Für jedes Software-Projekt wurde eine eigene Tabelle erstellt, in welcher neben der oben stehenden Metadaten zudem der Name des betreffenden Software-Projekts und die den Commits zugehörigen Release-Nummern gespeichert wurden. Jede veränderte Datei eines Commits erhält eine Zeile der Datenbank-Tabellen. In Tabelle XX kann eingesehen werden wie viele Releases je Software-Projekt zum Abruf einbezogen wurden und wie viele Commits daraus resultieren.

[3]Diese Information fand in der weiteren Erstellung des Datensets keine Verwendung.

Tabelle 3.3: Übersicht der zur Erstellung des Datensets verwendeten Software-Projekten mit zugehörigen Werten

	Zweck	Datenquelle	#Releases	#Commits	#Korrektiv	#Fehlereinführend	#Features
Blender	3D-Modellierungstool	GitHub-Mirror	11	19119	8258	1418	4637
Busybox	UNIX-Toolkit	Git-Repository	14	4984	1408	142	702
Emacs	Texteditor	GitHub-Mirror	7	12805	6959	685	863
GIMP	Bildbearbeitung	GitLab-Repository	14	7240	1703	272	1620
Gnumeric	Tabellenkalkulation	GitLab-Repository	8	6025	1591	136	725
gnuplot	Plotting-Tool	GitHub-Mirror	5	6619	880	1323	625
Irssi	IRC-Client	GitHub-Repository	7	253	77	1	17
libxml2	XML-Parser	GitLab-Repository	10	732	409	37	225
lighttpd	Webserver	Git-Repository	6	2597	1202	555	323
MPSolve	Polynomlöser	GitHub-Repository	8	668	158	69	130
Parrot	Virtuelle Maschine	GitHub-Repository	7	16245	3437	824	559
Vim	Texteditor	GitHub-Repository	7	9849	1033	2571	1227
xfig	Grafikeditor	Sourceforge-Repository	7	18	0	0	205

Diese „Rohdaten“ dienen zur weiteren Verarbeitung hinsichtlich der Erstellung des Datensets und der anschließenden Berechnung der Metriken. Eine Erläuterung der weiteren Verarbeitung der Daten folgt im kommenden Abschnitt.

3.2 Konstruktion des Datensets

Die Konstruktion des Datensets gliedert sich in mehrere Phasen der Datenverarbeitung und -optimierung. Die erste Phase besteht aus der Extraktion der involvierten Features einer veränderten Datei. Dazu wurden mithilfe eines Python-Scripts die sogenannten Präprozessor-Direktiven `#IFDEF` und `#IFNDEF` in den Diffs der veränderten Dateien identifiziert und anschließend die den Direktiven folgende Zeichenfolge bis zum Ende der Codezeile als Feature gespeichert. Die Identifizierung erfolgte mittels regulären Ausdrücken. Gespeichert werden die pro Datei identifizierten Features in einer zusätzlichen Spalte in den jeweiligen MySQL-Tabellen der Software-Projekte. Konnte kein Feature identifiziert werden, wird entsprechend `none` gepseichert.

Dieser Weg der Identifizierung birgt einige Hindernisse. Diese können, neben dem Normalfall, in Abbildung XX gesehen werden. In einigen C-Programmierparadigmen ist es üblich, Header-Dateien mittels Präprozessor-Direktiven in Sourcecode einzubinden, sodass sie wie Features scheinen (siehe erster unerwünschter Fall in Abbildung XX). Diese "Header-Features", wie sie im weiteren Verlauf genannt werden, sollten jedoch ignoriert werden, da sie im Sourcecode keine Variabilität erzeugen. In der Regel sind diese Header-Features identifizierbar durch ihre Namensgebung in Form eines angehängten `_h_` an den Featurenamen, wie beispielsweise `featurename_h_`. Dieser angehängte Teil erlaubt es, die Header-Features mittels regulärer Ausdrücke zu erkennen und auszufiltern.

Ebenfalls besteht die Möglichkeit, dass "falsche"Features identifiziert werden können. Beispiele dafür können von `#IFDEFs` stammen, welche in Kommentaren verwendet wurden (siehe zweiter unerwünschter Fall in Abbildung XX). Solche falschen Features wurden in einer manuellen Sichtung der identifizierten Features entfernt und durch `none` ersetzt.

Die nächste Phase der Verarbeitung besteht aus der Identifizierung von korrektiven Commits. Eine dafür gängige Methode, die auch in dieser Arbeit Anwendung fand, besteht aus der Analyse der Commit-Nachrichten auf das Vorhandensein von bestimmten Schlagworten (**HIER ANGABE ZU LITERATUR VON RODRIGO**). Bei den Schlagworten handelt es sich um „bug“, „error“, „fail“ und „fix“. Durchgeführt wurde die Analyse mittels Python-Skripte

```c
int test() {
       #IFDEF print_time
       printf("Current time: %s", time(&now));
       #ENDIF

       printf("Hello World!");
       return 0;
}
```

Normalfall
identifiziertes Feature: `print_time`

```c
#IFDEF time_h_
#INCLUDE <time.h>
#ENDIF

int test() {
       printf("Hello World!");
       return 0;
}
```

Unerwünschter Fall
identifiziertes Feature: `time_h_`
"Header-Features" werden ausgeschlossen

```c
// Maybe #IFDEF to make time optional?
int test() {
       printf("Current time: %s", time(&now));
       printf("Hello World!");
       return 0;
}
```

Unerwünschter Fall
identifiziertes Feature: `to make time optional?`
"falsche" Features werden manuell entfernt

Abbildung 3.1: Normalfall und unerwünschte Fälle bei der Identifizierung von Features

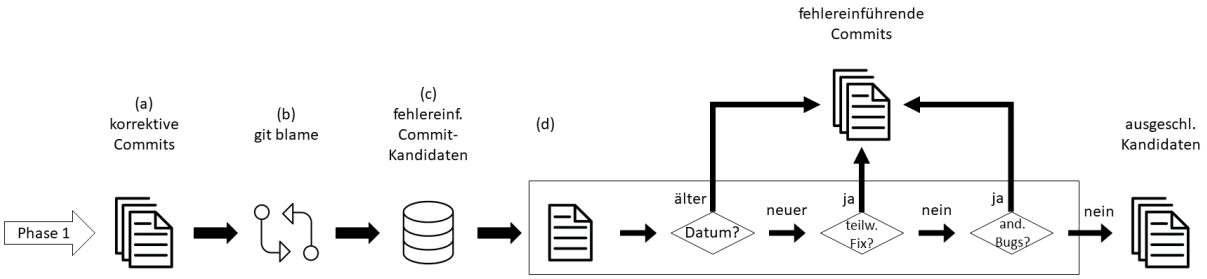

Abbildung 3.2: Ablauf der zweiten Phase des SZZ-Algorithmus (übersetzt, [**Borg2019**])

unter Zuhilfenahme von einfachen Formen des Text Minings. Die Ergebnisse wurden in einer weiteren boole'schen Spalte der MySQL-Tabellen (true = korrektiv, false = nicht korrektiv) gespeichert.

Der Suche nach korrektiven Commits folgt eine Analyse auf fehlereinführende Commits. Dazu wurde eine PyDriller-Implementierung des SZZ-Algorithmus nach Sliwerski, Zimmermann und Zeller verwendet [**Sliwerski2005**]. Dieser ursprünglich für CVS-Versionskontrollsysteme entwickelte Algorithmus erlaubt es, in zwei Phasen fehlereinführende Commits in lokal gespeicherten Software-Repositories zu finden [**Borg2019**]. Die erste Phase besteht dabei aus der Identifizierung der korrektiven Commits. Dies kann entweder anhand der zuvor beschriebenen Analyse der Commit-Nachrichten geschehen oder durch die Analyse von Bug-Tracking-Systemen [**Borg2019**]. Die zweite Phase umfasst die Identifikation der fehlereinführenden Commits auf Basis der zuvor erkannten korrektiven Commits. Diese Phase ist in mehrere Schritte unterteilt und wird in Abbildung XX dargestellt. Die Erläuterungen der mit Buchstaben versehenen Schritte erfolgt im Anschluss. Die PyDriller-Implementierung des Algorithmus folgt dem gezeigten Ablauf.

Die zweite Phase des SZZ-Algorithmus, die als Input eine Liste der Commit-Hashes der zuvor erkannten korrektiven Commits (a) erhält, beginnt mit der Ausführung eines `git blame` Befehls (b) zur Identifizierung sämtlicher Commits, in denen Veränderungen an den selben Dateien und Codezeilen vorgenommen wurden wie in den korrektiven Commits [**Borg2019**]. Dar-

Tabelle 3.4: Übersicht des Schemas der initialen MySQL-Tabellen

Spaltenname	Beschreibung	Spaltenname	Beschreibung
name	Name des Softwareprojekts	lines_added	Anzahl der hinzugefügten Zeilen zur geänderten Datei
release_number	zugehörige Release-Version basierend auf vergebenen Tags	lines_removed	Anzahl der entfernten Zeilen von der geänderten Datei
commit_hash	eindeutiger Bezeichner eines Commits	change_type	Art der Änderung
commit_author	Autor eines Commits	diff	Diff der geänderten Datei
commit_msg	Nachricht eines Commits	corrective	Indikator, ob Commit fehlerbehebend war
filename	Name der geänderten Datei	bug_introducing	Indikator, ob Commit fehlereinführend war
nloc	„Lines of code“ der geänderten Datei	feature	Namen der zugehörigen Features der geänderten Datei
cycomplexity	Zyklomatische Komplexität der geänderten Datei		

aus resultieren mögliche fehlereinführende Commit-Kandidaten (c). Für jeden dieser Commit-Kandidaten wird dann erörtert, ob er fehlereinführend ist (d). Dazu wird zunächst das Datum des Commit-Kandidaten mit dem zugehörigen korrektiven Commits verglichen. Liegt dieses vor dem Datum des korrektiven Commits, so gilt der Kandidat als tatsächlich fehlereinführend [**Borg2019**]. Liegt das Datum danach, so kann der Kandidat nur fehlereinführend sein, sofern er teilweise den vorhandenen Fehler löst (teilweiser Fix) oder für einen anderen Fehler verantwortlich ist, der nicht dem korrektiven Commit zugehörig ist (Kandidat ist Fehlerursache eines anderen korrektiven Commits) [**Borg2019**]. Die Ausgabe ist eine Liste von Commit-Hashes von fehlereinführenden Commits für jeden korrektiven Commit. Diese neuen Informationen werden in einer zusätzlichen boole'schen Spalte in den MySQL-Tabellen gespeichert (true = fehlereinführend, false = nicht fehlereinführend).

Eine Übersicht des Schemas der nun vollständigen initialen MySQL-Tabellen ist in Tabelle XX aufgeührt.

HIER!

3.3 Metriken

Ergänzen!!!!

Tabelle 3.5: Übersicht des Schemas der Metrics-Tabellen des Datensets

Spaltenname	Beschreibung	Spaltenname	Beschreibung
name	Name des Softwareprojekts	oexp	Ërfahrung"des Entwicklers, der am meisten zum betreffenden Feature / zur betreffenden Datei in einem Release beigetragen hat
release_number	zugehörige Release-Version basierend auf vergebene Tags	scat	Scattering Degree des betreffenden Features / der betreffenden Datei
feature / filename	betreffendes Feature / betreffende Datei	tang	Tangling Degree des betreffenden Features / der betreffenden Datei
comm	Anzahl der Commits, die in einem Release dem betreffenden Feature / der betroffenen Datei gewidmet sind	nloc	Durchschnittliche Lines of Code der Bearbeitungen des betreffenden Features / der betreffenden Datei in einem Release
adev	Anzahl der Entwickler, die das betreffende Feature / die betreffende Datei in einem Release bearbeitet haben	cyco	Durchschnittliche zyklomatische Komplexität der Bearbeitungen des betreffenden Features / der betreffenden Datei in einem Release
ddev	kummulierte Anzahl der Entwickler, die das betreffende Feature / die betreffende Datei in einem Release bearbeitet haben	addl	Durchschnittliche Anzahl der hinzugefügten Zeilen des betreffenden Features / der betreffenden Datei in einem Release
exp	Geometrisches Mittel der Ërfahrung" aller Entwickler, die am betreffenden Feature / an der betreffenden Datei in einem Release gearbeitet haben	reml	Durchschnittliche Anzahl der entfernten Zeilen des betreffenden Features / der betreffenden Datei in einem Release

Kapitel 4

Training und Test der Machine-Learning-Klassifikatoren

Ausblick: Dieses Kapitel gibt einen detaillierten Einblick in das Training der Machine-Learning-Klassifikatoren. Dazu werden zunächst die verwendeten Klassifikatoren und deren initiale Auswahl erläutert. Anschließend werden der Trainingsprozess sowie die zum Einsatz kommenden Softwarewerkzeuge beschrieben.

4.1 Auswahl der Werkzeuge und Klassifikationsalgorithmen

Durch die Wahl der Programmiersprache Python, war die Entscheidung zur Auswahl eines Machine-Learning-Werkzeugs bereits absehbar. Zur Anwendung kommt die Python-Library scikit-learn[1], die im Jahr 2007 von Pedregosa et. al entwickelt wurde [**scikit**]. Das Werkzeug bietet eine große Auswahl an Machine-Learning-Algorithmen für überwachtes und unüberwachtes Lernen und ermöglicht darüber hinaus eine einfache Implementation sowie eine einfache Einbindung weiterer Python-Libraries, wie beispielsweise die Matplotlib zur Erstellung von mathematischen Darstellungen [**scikit**].

Ebenfalls wird der WEKA-Workbench[2] als weiteres Machine-Learning-Wekzeug verwendet. Im Rahmen der strukturierten Literaturanalyse zu Beginn der Erarbeitung der Masterarbeit, erwies sich dieses Werkzeug durch zahlreiche Zitierungen in wissenschaftlichen Arbeiten (unter anderem in [**Queiroz2016**]) ebenfalls als geeignet. Der WEKA-Workbench (WEKA als Akronym für Waikato Environment for Knowledge Analysis) wurde an der University of Waikato in Neuseeland entwickelt und bietet eine große Kollektion an Machine-Learning-Algorithmen und Preprocessing-Tools zur Verwendung innerhalb einer grafischen Benutzeroberfläche [**Weka2016**].

Die Verwendung von zwei Machine-Learning-Werkzeugen ermöglicht einen Vergleich der jeweiligen Implementierungen der verwendeten Klassifikationsalgorithmen in der anschließenden Evaluation. Eine Übersicht über die ausgewählten Klassifikationsalgorithmen befindet sich in Tabelle XX. Kurze Erläuterungen der Algorithmen befinden sich im Anschluss.

[1] https://scikit-learn.org/
[2] https://www.cs.waikato.ac.nz/ml/weka/

Tabelle 4.1: Zum Training verwendete Klassifikationsalgorithmen

scikit-learn	**WEKA**
Decision Trees	J48-Decision-Trees
k-Nearest-Neighbors	k-Nearest-Neighbors
Ridge Classifier	Logistic Regression
Naive Bayes	Naive Bayes
künstliche neuronale Netze	künstliche neuronale Netze
Random Forest	Random Forest
Stochastic Gradient Descent	Stochastic Gradient Descent
Support Vector Machines	Support Vector Machines

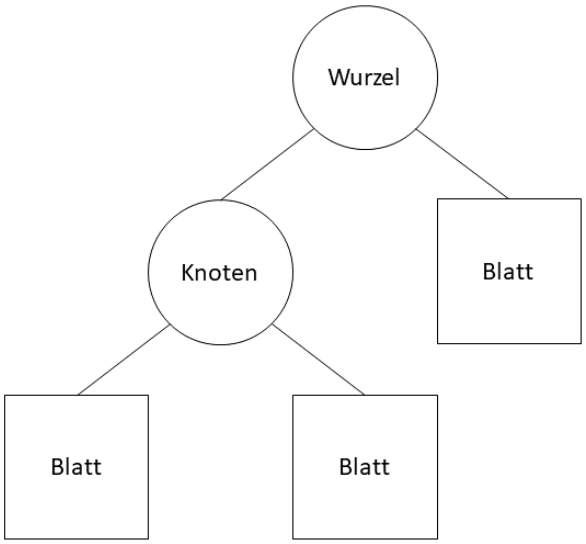

Abbildung 4.1: Grundsätzlicher Aufbau eines Decision Trees

Decision Trees

Überarbeiten?

Decision Trees (deutsch: Entscheidungsbäume) zählen zu den meistverwendeten Klassifikatoren im Bereich des supervised Machine Learnings. Studien belegten, dass sie hinsichtlich der Verwendung im Kontext von Fehlererkennung am häufigsten Anwendung finden [4]. Decision Trees sind gerichtete und verwurzelte Bäume, die als rekursive Partition der Eingabemenge des Datensets aufgebaut wird [**Rokach2005**]. Den Ursprung des Baumes bildet die Wurzel, welche keine eingehenden Kanten besitzt - alle weiteren Knoten besitzen jedoch eine eingehende Kante [**Rokach2005**]. Diese Knoten teilen wiederum die Eingabemenge anhand einer vorgegebenen Funktion in zwei oder mehr Unterräume der Menge auf [**Rokach2005**]. Meist geschieht dies anhand eines Attributs, sodass die Eingabemenge anhand der Werte des einzelnen Attributs geteilt wird [**Rokach2005**]. Die Blätter des Baumes bilden die Zielklassen ab. Eine Klassifizierung kann folglich durchgeführt werden, indem man von der Wurzel bis zu einem Blatt den Kanten anhand der entsprechenden Werte der Eingangsmenge folgt. Es existieren verschiedene Algorithmen zur Erstellung von Decision Trees. Bekannte Stellvertreter dieser sind ID3, C4.5 (J48) und CART [**Rokach2005**]. Der grundlegende Aufbau eines Decision Trees ist in Abbildung XX dargestellt.

Eine Besonderheit von Decision Trees stellen sogenannte Random Forests dar. Diese beschreiben eine Lernmethode von Klassifikatoren, bei der mehrere einzelne Decision Trees gleichzeitig

$$D(p,q) = \sqrt{\sum_{1}^{n} (p_n - q_n)^2}$$

Abbildung 4.2: Formel zur Berechnung der Euklidischen Distanz (n = Anzahl der Attribute)

erzeugt werden und deren Ergebnisse anschließen aggregiert werden [**Alam2013**]. Dazu erhält jeder Decision Tree eine Teilmenge der Eingabemenge des Datensets [**Alam2013**]. Random Forests eigenen sich besonders zur Anwendung, wenn viele Attribute im Datenset vorhanden sind [**Alam2013**].

k-Nearest-Neighbors

Ein k-Nearest-Neighbor-Klassifikator (deutsch: k-nächste-Nachbarn) basiert auf zwei Konzepten [**Zhang2016**]. Das erste basiert auf der Abstandsmessung zwischen den Werten der zu klassifizierenden Datenmenge und den Werten der Attribute des Datensets [**Zhang2016**]. Die Abstandmessung erfolgt in der Regel durch die Berechnung der Euklidischen Distanz (siehe Abbildung XX). Das zweite Konzept bildet der Parameter k, der angibt, wie viele nächste Nachbarn zum Vergleich der zuvor berechneten Abstände in Betracht gezogen werden. Bei einem k > 1 wird diejenige Zielklasse gewählt, deren Auftreten innerhalb der nächsten Nachbarn überwiegt.

Künstliche neuronale Netze

Künstliche neuronale Netze (KNN) verwenden nicht-lineare Funktionen zur schrittweisen Erzeugung von Beziehungen zwischen der Eingabemenge und den Zielklassen durch einen Lernprozess [**Linder2004**]. Sie sind angelehnt an die Funktionsweise von biologischen Nervensystemen und bestehen aus einer Vielzahl von einander verbundenen Berechnungsknoten, den Neuronen [**OShea2015**]. Der grundsätzliche Aufbau eines künstlichen neuronalen Netzes kann in Abbildung XX eingesehen werden. Der Lernprozess besteht aus zwei Phasen - einer Trainingphase und einer Recall-Phase [**Linder2004**]. In der Trainingsphase werden die Eingabedaten, meist als multidimensionaler Vektor, in den Input-Layer geladen und anschließend an die Hidden-Layer verteilt [**OShea2015**]. In den Hidden-Layers werden dann Entscheidungen anhand der Beziehungen zwischen den Eingabedaten und Zielklassen sowie die den Verbindungen zuvor zugewiesenen Gewichtsfaktoren getroffen [**Linder2004**]. **HIER!** [**Linder2004**]

HIER!

Naïve Bayes

Naïve-Bayes-Klassifikatoren zählen zu den linearen Klassifikatoren und basieren auf dem Satz von Bayes. Die Bezeichnung "naivërhält der Klassifikator durch die Annahme, dass Attribute der Eingabemenge unabhängig voneinander sind (diese Annahme wird häufig verletzt, dennoch erzielt der Klassifikator eine hohe Performanz) [**Raschka2014**]. Der Klassifikator gilt als effizient, robust, schnell und einfach implementierbar [**Raschka2014**]. Die zur Durchführung einer Klassifikation mittels Naïve Bayes benötigte Formel nach Thomas Bayes ist in Abbildung XX samt Erläuterung aufgeführt.

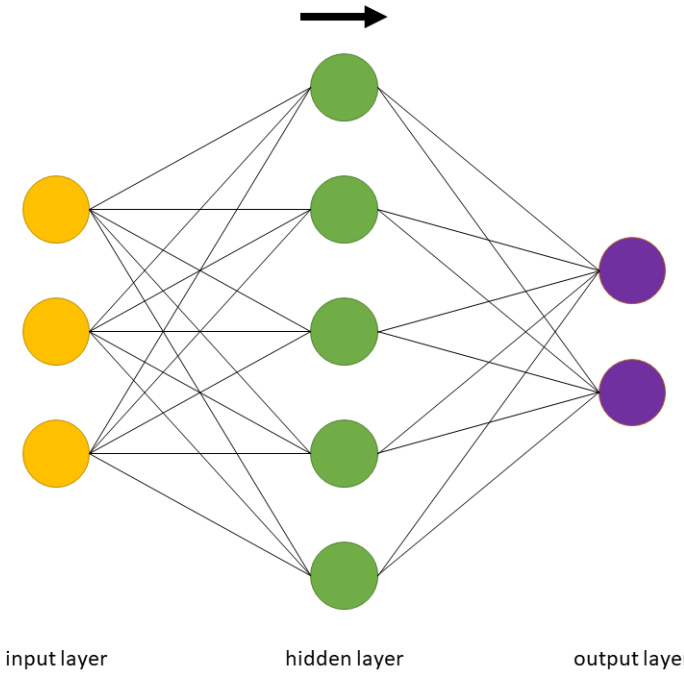

Abbildung 4.3: Grundsätzlicher Aufbau eines KNN mit 4 Input-Neuronen, 5 Hidden-Neuronen und 2 Output-Neuronen

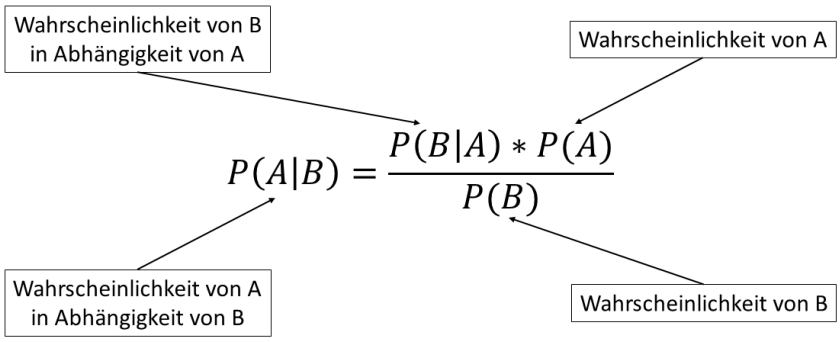

Abbildung 4.4: Satz von Bayes als Grundlage des Naïve-Bayes-Klassifikators

Es existiert zudem eine Mehrzahl an Varianten des Naïve-Bayes-Klassifikators, die verschiedene Annahmen über die Verteilung der Attribute der Eingabemenge machen. Beispiele dafür sind der Gauß'sche-Naïve-Bayes (normalverteilte Attribute), der multinomiale Naïve-Bayes (multinomiale Verteilung der Attribute) sowie der Bernoulli-Naïve-Bayes (unabhängige binäre Attribute).

Logistische Regression

Logistische Regressions-Klassifikatoren basieren auf dem mathematischen Konzept des Logits, welcher den natürlichen Logarithmus eines Chancenverhältnisses beschreibt [**Peng2002**]. Am besten geeignet ist dieser Klassifikator für eine Kombination aus kategorischen oder kontinuierlichen Eingabedaten und kategorischen Zielklassen [**Peng2002**].

HIER

Stochastic Gradient Descent

[**Bottou2010**]

Support Vector Machines

Support Vector Machines verfolgen das Ziel, linear separierbare Klassen [**Tzotsos2006**]

4.2 Analyse des Testprozesses

Kapitel 5

Evaluation

Ausblick: Dieses Kapitel dient der Evaluation der im vorangegangenen Kapitel erläuterten Klassifikationen. Dies geschieht durch verschiedene Vergleichsmetriken, welche in diesem Kapitel vorgestellt werden. Ebenfalls umfasst dieses Kapitel einen Vergleich der Klassifikatoren zu nicht-featurebasierten Methoden und eine Erläuterung der Herausforderungen und Limitationen, die mit der Erarbeitung der vorangegangenen Kapitel einhergingen.

5.1 Herausforderungen und Limitationen

5.2 Vergleich der Klassifikatoren

5.2.1 Vergleichsmetriken

5.2.2 Ergebnisse (TBD)

5.3 Vergleich zu nicht-featurebasierten Methoden

Kapitel 6

Fazit

Ausblick: Das abschließende Kapitel dieser Arbeit dient zur Zusammenfassung der Ergebnisse der vorangegangenen Kapitel sowie zur Erläuterung der daraus gewonnenen Erkenntnisse. Ebenfalls wird ein Ausblick auf eine mögliche Weiterführung dieser Arbeit gegeben.

6.1 Zusammenfassung und Erkenntnisse

6.2 Ausblick

Literatur

[1] Sven Apel u. a. *Feature-Oriented Software Product Lines*. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-37521-7.

[2] Venkata Udaya B. Challagulla u. a. „Empirical assessment of machine learning based software defect prediction techniques“. In: *International Journal on Artificial Intelligence Tools* 17.2 (2008), S. 389–400. ISSN: 02182130. DOI: 10.1142/S0218213008003947.

[3] Pete Chapman u. a. „CRISP-DM 1.0“. In: *CRISP-DM Consortium* (2000), S. 76. ISSN: 0957-4174. DOI: 10.1109/ICETET.2008.239.

[4] Le Son u. a. „Empirical Study of Software Defect Prediction: A Systematic Mapping“. In: *Symmetry* 11.2 (Feb. 2019), S. 212. DOI: 10.3390/sym11020212.

[5] Thomas Thüm u. a. „A Classification and Survey of Analysis Strategies for Software Product Lines“. In: *ACM Computing Surveys* 47.1 (Juni 2014), S. 1–45. DOI: 10.1145/2580950.

Anhang A

Links der für die Erstellung des Datensets verwendeten Software-Projekte

	Link zur Website	Link zum Repository
Blender	https://www.blender.org/	https://github.com/sobotka/blender
Busybox	https://busybox.net/	https://git.busybox.net/busybox/
Emacs	https://www.gnu.org/software/emacs/	https://github.com/emacs-mirror/emacs
GIMP	https://www.gimp.org/	https://gitlab.gnome.org/GNOME/gimp
Gnumeric	http://www.gnumeric.org/	https://gitlab.gnome.org/GNOME/gnumeric
gnuplot	http://gnuplot.info/	https://github.com/gnuplot/gnuplot
Irssi	https://irssi.org/	https://github.com/irssi/irssi
libxml2	http://www.xmlsoft.org/	https://gitlab.gnome.org/GNOME/libxml2
lighttpd	https://www.lighttpd.net/	https://git.lighttpd.net/lighttpd/lighttpd1.4.git/
MPSolve	https://numpi.dm.unipi.it/software/mpsolve	https://github.com/robol/MPSolve
Parrot	http://parrot.org/	https://github.com/parrot/parrot
Vim	https://www.vim.org/	https://github.com/vim/vim
xfig	https://sourceforge.net/projects/mcj/	https://sourceforge.net/p/mcj/xfig/ci/master/tree/

Websites zuletzt abgerufen am 13. Januar 2020.

Anhang B

Test 2

Lorem ipsum