*Masterthesis*

Featurebasierte Fehlervorhersage mittels Methoden des Machine Learnings

Stefan Hermann Strüder

8. November 2019

Universität Koblenz-Landau Fachbereich Informatik

[Arbeitsgruppe Software Engineering](http://ls14-www.cs.tu-dortmund.de/main/) – Prof. Dr. Jan Jürjens

betreut durch: Dr. Daniel Strüder

Stefan Hermann Strüder

stefanstrueder@uni-koblenz.de  
Matrikelnummer: 214200670  
Studiengang: Master Informatik

Masterthesis  
Thema: Featurebasierte Fehlervorhersage mittels Methoden des Machine Learnings

Eingereicht: 8. November 2019

Betreuer: Dr. Daniel Strüder

Prof. Dr. Jan Jürjens Arbeitsgruppe Software Engineering  
Fachbereich Informatik  
Universität Koblenz-Landau  
Universitätsstraße 1  
56070 Koblenz

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Sämtliche aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und noch nicht veröffentlicht.

Koblenz, den 8. November 2019 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Unterschrift

# Abstract

This is the abstract which is always written in English.

# Inhaltsverzeichnis

[Ehrenwörtliche Erklärung iii](#_Toc24122951)

[Abstract 4](#_Toc24122952)

[Inhaltsverzeichnis V](#_Toc24122953)

[Abbildungsverzeichnis VII](#_Toc24122954)

[Tabellenverzeichnis VIII](#_Toc24122955)

[1 Einleitung 1](#_Toc24122956)

[1.1 Forschungsziele und Forschungsfragen 3](#_Toc24122957)

[1.2 Forschungsdesign 4](#_Toc24122958)

[1.3 Zeitplanung 6](#_Toc24122959)

[1.4 Aufbau der Arbeit 6](#_Toc24122960)

[2 Hintergrund 7](#_Toc24122961)

[2.1 Featurebasierte Softwareentwicklung 7](#_Toc24122962)

[2.2 Machine-Learning-Klassifikation 7](#_Toc24122963)

[2.3 Fehlervorhersage mittels Machine Learning 7](#_Toc24122964)

[3 Entwicklung eines featurebasierten Datensets 8](#_Toc24122965)

[3.1 Datenauswahl 8](#_Toc24122966)

[3.2 Konstruktion des Datensets 8](#_Toc24122967)

[3.3 Metriken (oder 3.2 oder in ML Kapitel TBD) 8](#_Toc24122968)

[4 Anlernung von Machine-Learning-Klassifikatoren 9](#_Toc24122969)

[4.1 Auswahl der Klassifikatoren TBD 9](#_Toc24122970)

[4.2 Anlernungsprozesse TDB 9](#_Toc24122971)

[5 Evaluation 10](#_Toc24122972)

[5.1 Herausforderungen und Limitationen 10](#_Toc24122973)

[5.2 Vergleich der Klassifikatoren 10](#_Toc24122974)

[5.3 Vergleich zu nicht-featurebasierten Methoden 10](#_Toc24122975)

[6 Fazit 11](#_Toc24122976)

[6.1 Zusammenfassung und Erkenntnisse 11](#_Toc24122977)

[6.2 Ausblick 11](#_Toc24122978)

[Literaturverzeichnis 12](#_Toc24122979)

[Anhang 13](#_Toc24122980)

# Abbildungsverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

# Tabellenverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

# Einleitung

**ANPASSEN!**

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 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 *Features* (im Sinne von Charakteristika von Daten), um einen gegebenen Klassifikator zu anzulernen beziehungsweise zu trainieren. Dieser soll dann eine akkurate Vorhersage 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, dass innerhalb dieser Verfahren sowohl Entscheidungsbaum-basierte (zum Beispiel J48, CART oder Random Forest) als auch bayessche Verfahren die meistgenutzten sind (Son et al., 2019). Alternative Lernmethoden sind Regression, k-Nearest-Neighbor oder neuronale Netze (Challagulla, Bastani, Yen, & Paul, 2008). 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 Vorhersagetechnik für Softwarefehler basierend auf *Software-Features*. Diese beschreiben Inkremente der Funktionalität eines Softwaresystems. Die auf diese Weise entwickelten Softwaresysteme heißen *Software-Produktlinien*. Diese bestehen aus einer Menge von ähnlichen Softwareprodukten und zeichnen sich dadurch aus, dass sie eine gemeinsame Menge von Features sowie eine gemeinsame Codebasis besitzen (Thüm, Apel, Kästner, Schaefer, & Saake, 2014). Durch das Vorhandensein verschiedener Features entlang der Softwareprodukte, kann eine breite Variabilität innerhalb einer Produktlinie erreicht werden.

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 (Apel, Batory, Kästner, & Saake, 2013). 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.

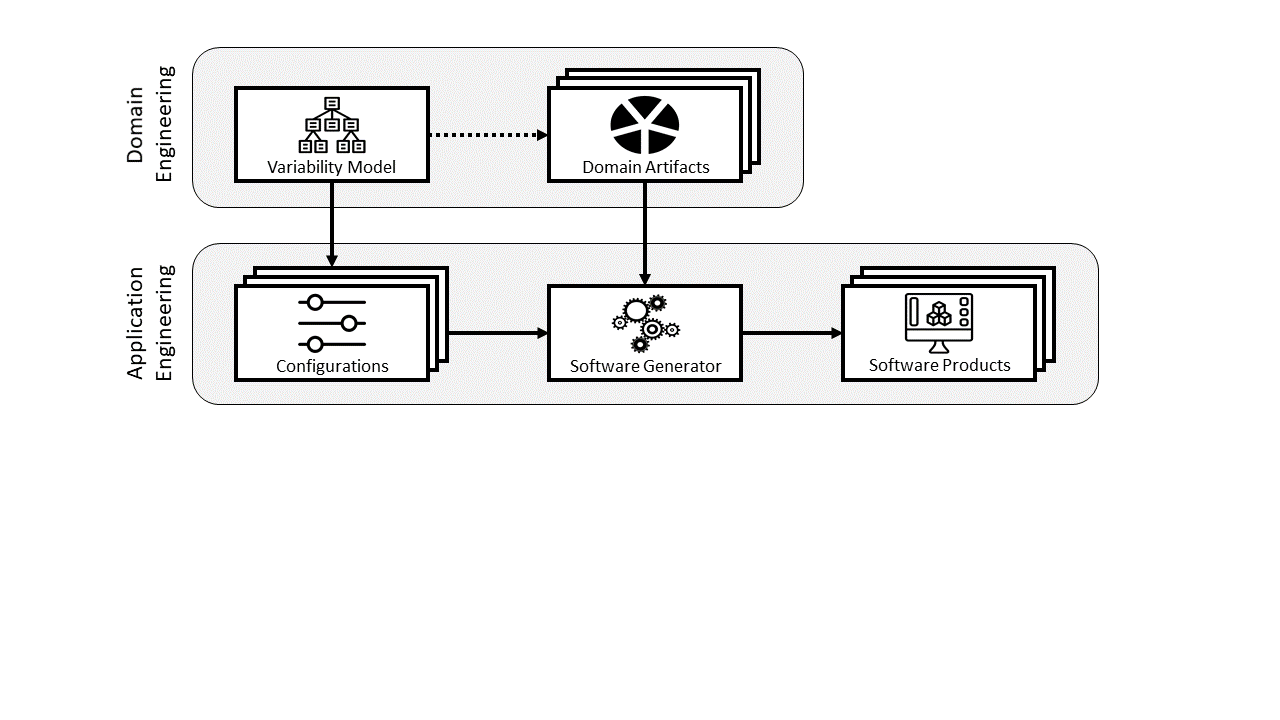


Abbildung 1.1: Generierung von Software-Produktlinien (nach Thüm et al., 2014)

Das Ziel dieser Arbeit ist es, eine auf Machine Learning gestützte Vorhersagetechnik unter Verwendung von Software-Features zu entwickeln. Diese Idee 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.

Dieses Ziel 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 in Kapitel 2.1.

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

## 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 Forschungszielen gehören eine Vor- und Nachbereitung der Arbeit, sodass sich insgesamt fünf Arbeitsphasen ergeben. Diese werden in den weiteren Unterkapiteln näher erläutert. Als finale Vorhersagetechnik wird jener Klassifikator verwendet, der im Rahmen der Gegenüberstellung während der Evaluation als am effektivsten hervorgeht.

## Forschungsdesign

**ANPASSEN!**

Die für diese Arbeit gewählte Methodik basiert auf dem Prozessmodell *Cross-Industry Standard Process for Data Mining*, kurz *CRISP-DM*, nach Chapman et al. (2000). Es wird als Vorlage für die Arbeitsphasen für die Erreichung der Forschungsziele dieser Arbeit verwendet. Da der überwiegende praktische Teil dieser Arbeit durch Programmierung im Bereich des Machine Learning geschieht, bildet das CRISP-DM Prozessmodell ein passendes vordefiniertes Vorgehen. Dieses wurde speziell für die Erarbeitung von *Data Mining* Projekten entwickelt, eignet sich jedoch auch für die Verwendung im Rahmen des Machine Learnings, 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 2.1 dargestellt. Zusätzlich umfasst die Abbildung die Zuordnung der Arbeitsphasen. Einen genauen Überblick über den konkreten Umfang der Arbeitsphasen bietet das im Anschluss folgende Unterkapitel 2.3.

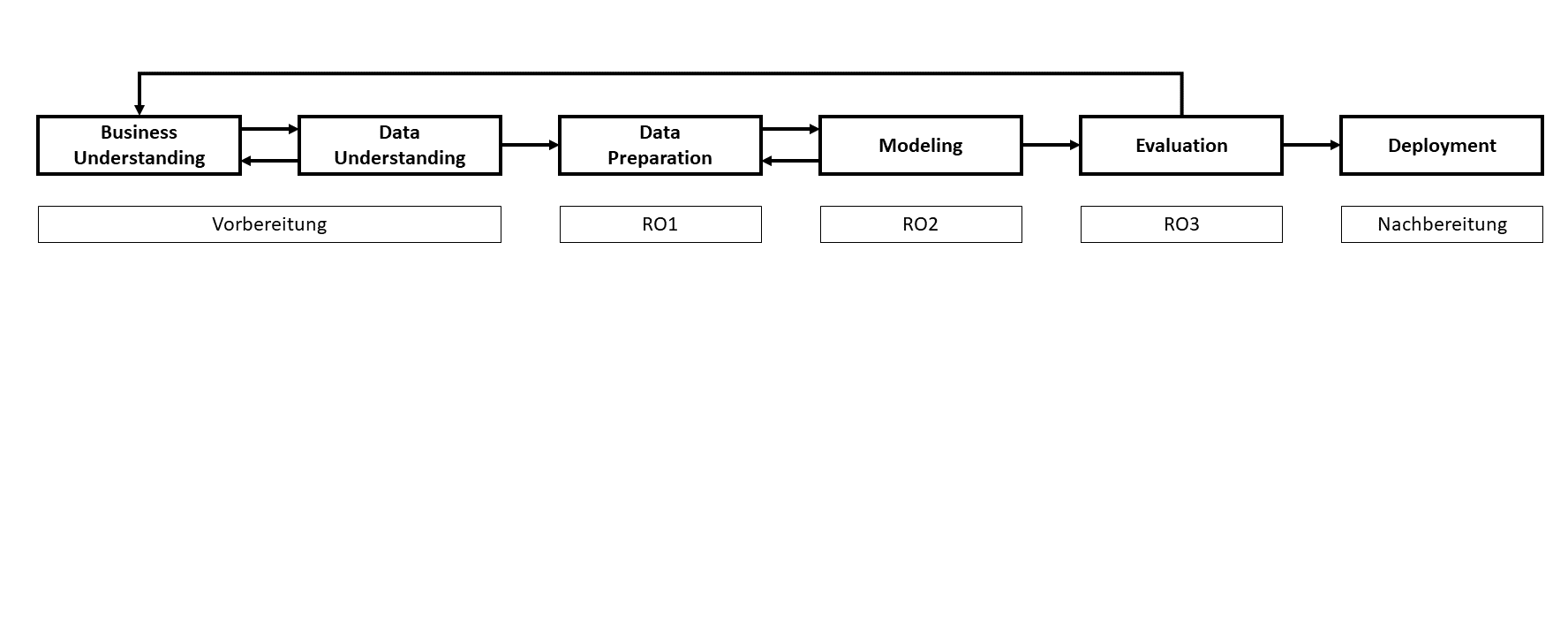


Abbildung 2.1: Phasen des CRISP-DM Prozessmodells (Chapman et al., 2000) mit Zuordnung der Arbeitsphasen

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 Methodik festgelegt (siehe Unterziele der ersten Phase im kommenden Abschnitt).

## 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** | **Dauer** |  |
| Strukturierte Literaturrecherche   * Techniken der featurebasierten Softwareprogrammierung * Techniken zur Fehlererkennung in Software * Klassifikation mittels Machine Learning * Klassifikationsmethoden * Auswahl der Programmiersprache * Tool- & Libraryauswahl * Evaluationsmetriken | 2 Wochen | 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 / Datensets | 1 Woche | Data Understanding |
| Total: | 3 Wochen |  |

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

|  |  |
| --- | --- |
| **Unterziele** | **Dauer** |
| finale Datenauswahl   * Festlegung von Kriterien | 1 Woche |
| Datenbereinigung   * „Preprocessing“ | 1 Woche |
| 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 |
| Total: | 3 Wochen |

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

|  |  |
| --- | --- |
| **Unterziele** | **Dauer** |
| Auswahl geeigneter Klassifikatoren | 1 Woche |
| Training der Klassifikatoren | 3 Wochen |
| Total: | 4 Wochen |

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

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

Phase 5: Nachbereitung (Deployment)

|  |  |
| --- | --- |
| **Unterziele** | **Dauer** |
| Besprechung der vorangegangenen Arbeit mit Betreuer   * Umsetzung möglicher Verbesserungsvorschläge | 1 Woche |
| Erstellung der Ausarbeitung\* | 7 Wochen |
| Erstellung der Abschlusspräsentation | 1 Woche |
| Total: | 9 Wochen |

\* 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.

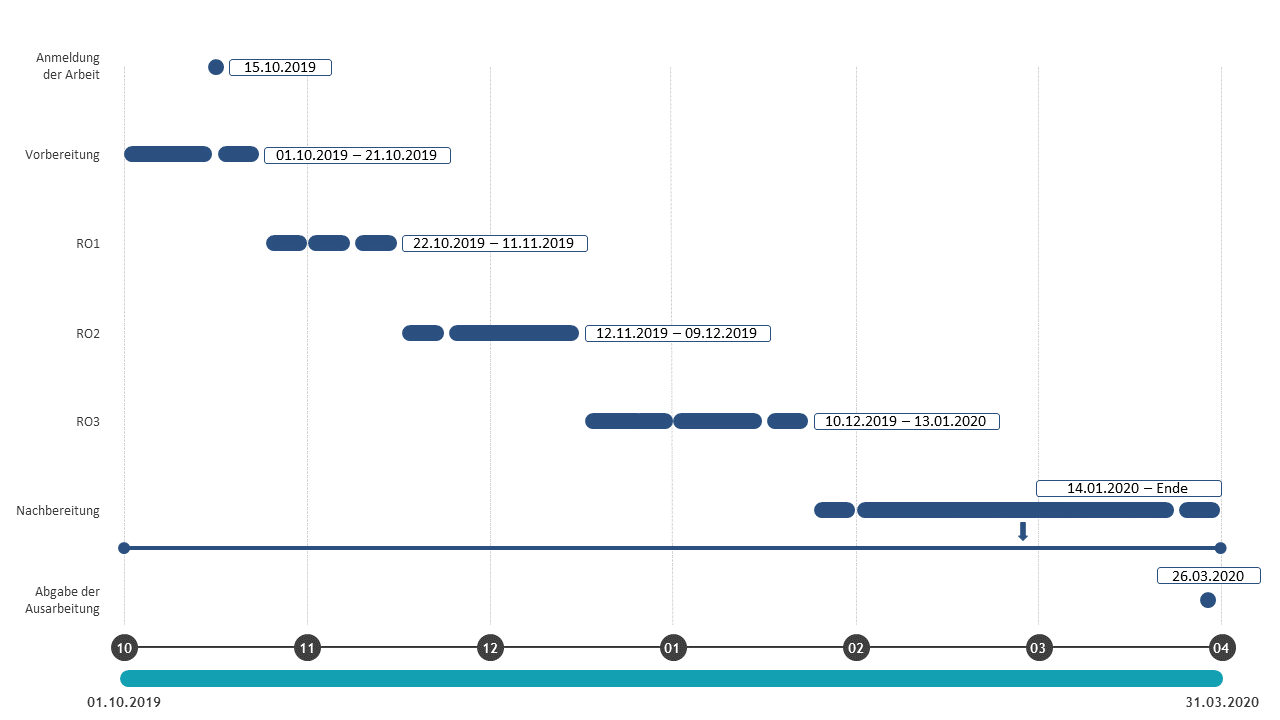


Abbildung 2.2: Zeitlicher Ablaufplan der Arbeit als Gantt-Chart

## Aufbau der Arbeit

# Hintergrund

## Featurebasierte Softwareentwicklung

## Machine-Learning-Klassifikation

## Fehlervorhersage mittels Machine Learning

# Entwicklung eines featurebasierten Datensets

## Datenauswahl

## Konstruktion des Datensets

## Metriken (oder 3.2 oder in ML Kapitel TBD)

# Anlernung von Machine-Learning-Klassifikatoren

## Auswahl der Klassifikatoren TBD

## Anlernungsprozesse TDB

# Evaluation

## Herausforderungen und Limitationen

## Vergleich der Klassifikatoren

## Vergleich zu nicht-featurebasierten Methoden

# Fazit

## Zusammenfassung und Erkenntnisse

## Ausblick

# Literaturverzeichnis

Apel, S., Batory, D., Kästner, C., & Saake, G. (2013). *Feature-Oriented Software Product Lines*. https://doi.org/10.1007/978-3-642-37521-7

Challagulla, V. U. B., Bastani, F. B., Yen, I. L., & Paul, R. A. (2008). Empirical assessment of machine learning based software defect prediction techniques. *International Journal on Artificial Intelligence Tools*, *17*(2), 389–400. https://doi.org/10.1142/S0218213008003947

Son, L. H., Pritam, N., Khari, M., Kumar, R., Phuong, P. T. M., & Thong, P. H. (2019). Empirical study of software defect prediction: A systematic mapping. *Symmetry*, *11*(2). https://doi.org/10.3390/sym11020212

Thüm, T., Apel, S., Kästner, C., Schaefer, I., & Saake, G. (2014). A classification and survey of analysis strategies for software product lines. *ACM Computing Surveys*, *47*(1). https://doi.org/10.1145/2580950

# Anhang