Identifikation von Quellcodestellen mit hoher Fehlerwahrscheinlichkeit durch maschinelle Lernverfahren

MASTERARBEIT

Robert Klinger

30. November 2021

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik
Professur für Open-Source-Software

Betreuer
Prof. Dr. Dirk Riehle, M.B.A.
Dr. Martin Jung
Dr. Niko Pollner

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 30. November 2021

License

This work is licensed under the Creative Commons Attribution 4.0 International license (CC BY 4.0), see https://creativecommons.org/licenses/by/4.0/

Erlangen, 30. November 2021

Abstract

Early identification of bugs in the software development process is an important aspect of modern development. This thesis examines correlation analyses between quantified software artifacts and bug occurrence using machine learning methods. An arc42-standard compliant documented open-source frameworks as a basis for researching statistical correlations of quantified software artifacts and their bugs as well as a basis for developing bug-identification programs has been published. A plugin was developed to support SonarQube to quantify files of git commits. SonarQube is able to measure various software metrics for up to 29 programming languages. For the annotation of bug-fixing commits the commit message and the changed files have been analyzed. Files from 10,000 commits of the TypeScript open source project were quantified and annotated. 233 features over the last five changes of a file were taken into account to predict the number of bug fixes of the next five changes of that file. Several machine learning techniques were tested including automated machine learning with TPOT and H2O. One model was able to predict with an accuracy of 83%. The average absolute error was 0.18. The weighted average of the F1-Scores was 81% and the unweighted was due to low support of the classes three and four 42%. All classes above one were set to one and binary classifiers have been developed. The best ones achieved an AUC of the ROC curve of 0.89.

Zusammenfassung

Die frühzeitige Identifikation von Fehlern im Softwareentwicklungsprozess ist ein wichtiger Bestandteil moderner Entwicklung. Diese Arbeit befasst sich mit Korrelationsuntersuchungen zwischen quantifizierten Softwareartefakten und Fehlerauftreten mithilfe maschineller Lernverfahren. Mit dem Ziel der besseren Teilung von Forschungsergebnissen veröffentlicht diese Arbeit ein nach Arc42 standardkonform dokumentiertes Open-Source-Framework als Basis für weitere Forschungen und für die Entwicklung Fehler identifizierender Programme. Für die Quantifizierung, von Dateien eines Git-Commits, wurde ein Plugin zur Unterstützung von SonarQube entwickelt. SonarQube ist in der Lage diverse Softwaremetriken für bis zu 29 Programmiersprachen zu messen. Zur Annotation Fehler-behebender Commits wurde die Commit-Nachricht und die geänderten Dateien analysiert. Es wurden exemplarisch Dateien aus 10.000 Commits des Open-Source-Projekts TypeScript quantifiziert und annotiert. Es wurden 233 Features über die letzten fünf Änderungen einer Datei berücksichtigt, um die Anzahl der Fehlerbehebungen der nächsten fünf Änderungen vorherzusagen. Dieser Lerndatensatz wurde veröffentlicht. Verschiedene Verfahren des maschinellen Lernens wurden getestet, darunter automatisches maschinelles Lernen mit TPOT und H2O. Ein Modell konnte mit einer Accuracy von 83% die richtige Klasse vorhersagen, der durchschnittliche absolute Fehler betrug 0,18, der gewichtete Durchschnitt der F1-Scores betrug 81%, der ungewichtete betrug, aufgrund des geringen Supports der Klassen drei und vier, 42%. Es wurden alle Klassen über eins auf eins gesetzt und binäre Klassifikatoren entwickelt. Die besten erzielten eine AUC der ROC-Kurve von 0,89.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

z.B. zum Beispiel

z.T. zum Teil

bzw. beziehungsweise

PCA Principal Component Analysis

TPOT Tree-based Pipeline Optimization Tool

DTC Decision Tree Classifier

RFC Random Forest Classifier

GBC Gradient Boosting Classifier

LR Logistic Regression

ROC Receiver Operating Characterstic

AUC Area Under the Curve

1 Einleitung

Fehler während der Entwicklung von Softwareprodukten stellen einen Kostenfaktor und ein Sicherheitsrisiko dar (Shin et al., 2010). Erfahrungsgemäß gilt: Je später Fehler identifiziert werden, desto teurer ist deren Behebung. Erfolgt die Identifikation eines Fehlers spät, sind häufig mehrere Komponenten des Produkts betroffen. Die Identifikation sowie die Behebung der betroffenen Komponenten erfordert dadurch in aller Regel im fortgeschrittenen Verlauf der Entwicklung einen höheren Aufwand als zum Zeitpunkt der Erzeugung des Fehlers. Naheliegenderweise werden diverse Strategien zur frühzeitigen Vermeidung und Behebung von Fehlern als Teil des Qualitätssicherungsprozesses verfolgt. Besonders das Testen von Software hat sich als wichtige Maßnahme zur Sicherung einer geeigneten Zuverlässigkeit etabliert. Eine besondere Rolle in Bezug auf die Qualitätssicherung spielt die Softwarearchitektur. „[...] Entwurfsentscheidungen, die innerhalb der Softwarearchitektur getroffen werden, unterstützen oder behindern die Erreichung nahezu aller Qualitätsmerkmale einer nicht-naheliegenderweise Anwendung.“ (Clements et al., 2002, S. 11) Die Entwicklung vorteilhafter Softwarearchitekturen kann demnach zur Vermeidung von Fehlern beitragen. Je nach Definition, wird die Qualität der Softwarearchitektur als Erfüllungsgrad der Wünsche und Bedürfnisse eines Kunden beziehungsweise Benutzers oder als genaue und messbare Variable, mit welcher eine Rangordnung zwischen verschiedenen Produkten hergestellt werden kann, spezifiziert (Malich, 2008, S. 50f.). Beide Definitionen stellen einen domänenspezifischen Kontext her: Die Bedürfnisse des Kunden beziehungsweise Benutzers oder bestimmte Produkte. Die Bewertung von Softwarearchitekturen ist deshalb domänenspezifisch und wird aus pragmatischen Gründen, wenn überhaupt, nur in Teilen automatisiert. Eine Möglichkeit, einen Teil des Entwurfs und der Implementierung dennoch quantitativ zu bewerten, sind Softwarequalitätsmetriken. „Eine Softwarequalitätsmetrik ist eine Funktion, die eine Software-Einheit in einen Zahlenwert abbildet, welcher als Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit interpretierbar ist.“ („IEEE Standard for a Software Quality Metrics Methodology“, 1994, S. 3, übersetzt) Die Berechnung und Auswertung solcher Werte könnte eine geeignete Vorgehensweise zur Identifikation kritischer Stellen sein. Erste Arbeiten indizieren Potenziale der Auswertung von Softwaremetriken: Es gelang über 80 % der Dateien im Mozilla Firefox Web Browser

sowie im Red Hat Enterprise Linux Kernel, bei denen Sicherheitslücken bekannt waren, mit einer Falsch-Positiv-Rate von weniger als 25 % durch eine solche Auswertung zu identifizieren (Shin et al., 2010). In einem Java-Legacy-System konnte mithilfe eines Modells zur Vorhersage der Klassen-Fehler-Disposition mit einer Falsch-Positiv- und Falsch-Negativ-Rate von weniger als 20% eine geschätzte potenzielle Ersparnis des Aufwands des Verifikationsprozesses von 29% erreicht werden (Arisholm & Briand, 2006). In zwei industriellen Systemen enthielten 20% der Dateien mit der höchsten vorhergesagten Fehlerdichte im Schnitt 83% aller Fehler (Ostrand et al., 2005). Trotz einiger publizierter Forschungsarbeiten zur automatischen Auswertung von Softwaremetriken ist mir kein ausreichend generisches, in Hinblick der Begutachtung unterschiedlicher Stellen, und dokumentiertes Werkzeug zur anwendungsorientierten Nutzung zur Identifikation von Quellcodestellen mit hoher Fehlerwahrscheinlichkeit bekannt. Ich vermute die Komplexität der Aufgabe, eine unzureichend offen lizenzierte und gut auffindbare Publizierung wichtiger Teile der Forschungsergebnisse, wie Lerndatensatzbänke, Quantifizierungsprogramme und passende (trainierte) Modelle des maschinellen Lernens als hemmende Faktoren bezüglich der Entwicklung eines solchen Tools. Diese Arbeit soll all diese Punkte adressieren.

1.1 Thematischer Beitrag und Abgrenzung

Diese Arbeit beschäftigt sich mit der Automatisierung der Ermittlung und Auswertung von Softwaremetriken.
Es soll eine Open-Source-konform lizenzierte Basis für weitere Forschungsarbeiten und Produkte auf der weit bekannten Webseite www.github.com publiziert werden. Darunter fällt die Entwicklung eines Frameworks zur automatischen Ermittlung von Softwaremetriken, einer Standard-konformen Dokumentation dieses Frameworks, einen Klassifikator, welcher Softwaremetriken auf das (nicht-) vorhanden-sein eines Fehlers abbildet, diese Arbeit, in welcher insbesondere ein Klassifikator exemplarisch beschrieben und evaluiert wird, und eine exemplarische Datenbank mit Messungen zu Softwareartefakten und Annotationen, welche eine hohe beziehungsweise niedrige Fehlerwahrscheinlichkeit indizieren.

Abgrenzung zu verwandten Arbeiten: (Arisholm und Briand (2006), Ostrand et al. (2005), Shin et al. (2010), Liu et al. (2018), Ostrand und Weyuker (2010), Weyuker et al. (2008))

1. Es wird ein Lerndatensatz mit Softwaremetriken von Quellcode und Annotationen zur Fehlerbehaftung veröffentlicht.
2. Es wird ein Standard-konform dokumentiertes Framework publiziert, welches weitere Forschungen, mit dem Ziel der Entwicklung eines ausreichend generischen Werkzeugs, hinsichtlich der Art des zu analysierenden Projektes,

zur automatischen Fehleridentifikation von Softwareartefakten (allgemein: Lokalitäten), wie beispielsweise Quelldateien unterschiedlicher Programmiersprachen, vereinfachen soll.

3. Die Art der Stelle, die quantifiziert, annotiert und ausgewertet werden soll, wird offen gehalten. Insbesondere werden Vorhersagen für Releases, Commits, einzelne Dateien oder andere Stellentypen ermöglicht.

4. Es wird exemplarisch untersucht, inwieweit automatisiertes maschinelles Lernen geeignet ist, Fehler anhand gemessener Softwaremetriken zu identifizieren.

Nicht thematisiert wird, ob Veränderungen während der Entwicklung, mit dem Ziel einer Verbesserung der Softwaremetrik-Werte, die Fehler-Auftrittswahrscheinlichkeit reduzieren. Ebenfalls wird nicht die Generalisierbarkeit der errechneten statistischen Vorhersagemodelle auf andere Projekte untersucht.

1.2 Aufbau dieser Arbeit

Diese Arbeit ist in sechs Kapitel unterteilt. Ersteres ist bekannt, im zweiten Kapitel werden die wichtigsten Ergebnisse verwandter Arbeiten, welche zum Einsatz kommen, zusammengefasst. Sie dienen insbesondere als Basis für die Konzepte der Fehler-Annotation von Lokalitäten, der Findung geeigneter Modelle zur Prädiktion (Vorhersage) von Fehlern und Verfahren des automatischen maschinellen Lernens.

Kapitel drei behandelt Grundlagen, auf welche diese Arbeit aufbaut. Darunter fallen Konzepte und Algorithmen des maschinellen Lernens, welche benötigt werden, um einen Klassifikator zu entwickeln und zu evaluieren. Es werden die verwendeten Technologien und Konzepte der Implementierung der Komponenten sowie Grundlagen der Architekturdokumentation dieser Arbeit erläutert. Insbesondere wird Arc42 gezeigt, welches als Grundlage für das Kapitel vier dient. Zur Erreichung der Austauschbarkeit und Erweiterbarkeit des Projektes wird Dependency Injection, SonarQube als Werkzeug zum Messen von Softwaremetriken und MongoDB für die Persistierung eingeführt.

Im vierten Kapitel wird das Framework, inklusive einiger Plugin-Implementierungen, orientiert am Arc42-Template, dokumentiert. Es werden Anforderungen, der Ablauf einzelner Schritte zur Erzeugung eines Lerndatensatzes für das maschinelle Lernen, die statische Dekomposition des Projektes, die Verteilung der Komponenten auf Umgebungen sowie Konzepte zur Erreichung wichtiger qualitativer Anforderungen dokumentiert und modelliert. Dieses Kapitel soll Stakeholdern helfen einen Überblick über das Framework zu erlangen.

Das Kapitel fünf beschreibt, wie mit der in Kapitel vier beschriebenen Software, Daten erhoben wurden und Verfahren des maschinellen Lernens angewandt wur-

den. Es werden die erhobenen Daten und das Vorgehen der Verarbeitung dieser beschrieben. Es wird das durchgeführte Vorgehen der Erzeugung der Modelle, die Modelle selbst und deren Evaluation, gezeigt. Ebenfalls wird die Anwendung automatischer maschineller Lernverfahren präsentiert und im bestehenden Kontext evaluiert.

Das Kapitel sechs fasst diese Arbeit kurz zusammen und beschreibt perspektivisch weitere Untersuchungen und Implementierungen, welche wünschenswert sind und nicht durchgeführt wurden.

2 Verwandte Arbeiten

Die Prädiktion von Fehlern wurde in diversen Arbeiten untersucht. Diese schlagen Strategien und geeignete Modelle zur Fehlerprädiktion vor und zeigen ungewünschte Einschränkungen dieser Systeme auf. Andere Arbeiten zeigen die Möglichkeit automatisierten maschinellen Lernens und Möglichkeiten zur Identifikation korrektiver Commits.

2.1 Fehlervorhersage

Dieses Kapitel behandelt ausgewählte Arbeiten, welche sich das Ziel gesetzt haben Fehler in Quellcode zu entdecken. Diese Arbeiten verwendeten zur Erhebung der Features verschiedene Tools und Systeme. Alle betrachteten Arbeiten waren eingeschränkt auf bestimmte Programmiersprachen und/oder projektspezifische Systeme, wie ein Fehler-Tracker-System oder eine Fehler-Datenbank (Arisholm und Briand, 2006, Ostrand et al., 2005, Shin et al., 2010, Ostrand und Weyuker, 2010). Solche Einschränkungen begrenzen die Anwendungsmöglichkeiten eines automatisierten Fehleridentifikationsprogramms. Diese Arbeit vermeidet Anforderungen dahingehend besser, dass eine höhere Bandbreite verschiedenartiger Projekte untersucht werden kann. Es werden mit SonarQube viele Programmiersprachen unterstützt und die einzige weitere Anforderung ist, dass das betrachtete Projekt ein git-Repository ist. Das entwickelte Framework hält einschränkendere Entwicklungen durchaus offen.

Als funktionierende Modelle wurden die Negativ Binomiale Regression (Arisholm und Briand, 2006, Ostrand et al., 2005), die Logistische Regression (Arisholm und Briand, 2006), ein Random Forest Klassifikator, welcher in einem Fall ähnlich gut wie die Negativ Binomiale Regression abschnitt (Ostrand und Weyuker, 2010), genannt. Die logistische Regression und der Random Forest Klassifikator werden auch in dieser Arbeit verwendet.

Vier der betrachteten Arbeiten sagten auf Basis vergangener Releases und Änderungen zu diesen Fehler/Fehlerwahrscheinlichkeiten aller Dateien eines neuen Releases vorher (Arisholm und Briand, 2006, Ostrand et al., 2005, Shin et al.,

2010, Ostrand und Weyuker, 2010). Diese Arbeit versucht auf Basis vergangener Änderungen Fehler in nachfolgenden Änderungen vorherzusagen. Aufgrund der unterschiedlichen Quantifizierungsstrategien ist die Eignung der vorgeschlagenen Modelle nicht gesichert.

Eine Arbeit zeigte die erfolgreiche Möglichkeit der Fehlervorhersage anhand von Abstract-Syntax-Tree- (AST) Analysen von Java-Quellcode auf (Liu et al., 2018). Eine Arbeit zeigte die erfolgreiche Prädiktion anhand von Netzwerkanalysen über die Entwickleraktivitäten an den Dateien auf (Shin et al., 2010). Zukünftigen Arbeiten wird die Untersuchung möglicher Ensemble-Verfahren, Verwendung mehrerer Modelle zur Vorhersage (siehe Kapitel 3), dieser Arbeit und solcher Verfahren zur Steigerung der Klassifikationsgüte überlassen. Einschränkungen auf spezifische Programmiersprachen sind im Falle von AST-Analysen wahrscheinlich die Folge.

2.2 Identifikation korrektiver Commits

Zur Unterstützung überwachter Lernverfahren wird ein annotierter Datensatz benötigt. Anhand der Schlüsselbegriffe „bug, error, fixup, fail“ sollen korrektive Commits identifiziert werden können (Mockus & Votta, 2000). Commits, welche mehr als zwei Dateien betreffen, werden nicht als korrektiv klassifiziert (Ostrand et al., 2005, S. 5, Daumenregel).

2.3 Automatisiertes maschinelles Lernen

Es gibt mehrere Arbeiten für automatisiertes maschinelles Lernen. Die folgenden zwei Bibliotheken werden in dieser Arbeit verwendet und evaluiert.

TPOT, „Tree-bases Pipeline Optimization Tool“ (Le et al., 2020) (Baum-basiertes Pipeline-Optimierungs-Werkzeug), ist ein Tool zur automatischen Erzeugung und Optimierung von Pipelines für das maschinelle Lernen unter Verwendung der evolutionären Programmiertechnik Genetische Programmierung (Le et al., 2020).

„H2O ist eine quelloffene, verteilte Plattform für das maschinelle Lernen, designt, um für sehr große Datenmengen zu skalieren, mit APIs in R, Python, Java und Scala.“ (LeDell und Poirier, 2020, übersetzt)

3 Grundlagen

In diesem Kapitel werden ausgewählte Themen der Informatik, die zur Zielerreichung in den fortführenden Kapiteln genutzt werden, kurz erläutert.

3.1 Maschinelles Lernen

„Maschinelles Lernen ist das Studium von Computer-Algorithmen, welche sich automatisch durch Erfahrung verbessern. Applikationen reichen von Data-Mining-Programmen, welche generelle Regeln in großen Datensätzen entdecken, bis zu Informations-Filter-Systemen, die automatisch Nutzerinteressen erlernen.“ („IEEE SMC Maschinelles Lernen“, 2020, übersetzt)

Mittels Algorithmen aus dem Bereich des maschinellen Lernens können Prädiktionen (Vorhersagen) getroffen werden. Von besonderem Interesse ist die automatische Vorhersage von Fehlerstellen in Quellcode. In dem hier zu untersuchenden Ansatz werden große Mengen an bestehendem Quellcode analysiert und Zusammenhänge zwischen diesen und dem Auftreten von Fehlern untersucht. Es handelt sich also um einen Data-Mining-Prozess.

3.1.1 Klassifikation und Regression

Im Falle der Klassifikation wird eine Menge von Variablen auf einen diskreten Wertebereich, im Falle der Regression auf einen beliebigen Wertebereich, abgebildet.
Beispiel: Zur Erkennung von Hunden auf Bildern soll ein Programm (Klassifikator) Bilder in Bilder mit Hunden und Bilder ohne Hunde unterteilen. Es handelt sich um einen binären Klassifikator, welcher alle Bilder in den diskreten Wertebereich {Hund, kein Hund} abbildet.
Im Falle der Regression wird eine Menge von Variablen auf einen nicht diskreten, in der Praxis sehr großen, Wertebereich, abgebildet. Dies dient der Vorhersage bestimmter Werte eines großen Wertebereichs, z.B. zur Bestimmung von Aktienkursen in der Zukunft.

3.1.2 Überwachtes Lernen

Beim überwachten Lernen (supervised learning) werden einem Algorithmus in der Lernphase sowohl Eingaben (unabhängige Variablen) als auch die dazugehörigen, korrekten Ausgaben (abhängige Variablen) zur Verfügung gestellt. Durch das Lernen wird ein statistisches Modell erzeugt. In der Phase der Inferenz, also der Anwendung des statistischen Modells, stehen offensichtlicher Weise die korrekten Ausgaben nicht mehr zur Verfügung.
Beispiel: Anhand der Körpergröße und dem Körpergewicht soll das statistische Risiko einen Herzinfarkt im nächsten Jahr zu erleiden, vorhergesagt werden. Beim überwachten Lernen werden dem Algorithmus für die Lernphase Daten über Personen übergeben, bei denen die Körpergröße, das Gewicht und die Tatsache, ob innerhalb eines Jahres nach Messung dieser Attribute ein Herzinfarkt aufgetreten ist, übergeben. Anschließend kann das trainierte Modell genutzt werden, um Prognosen zum Auftreten eines Herzinfarktes im nächsten Jahr, anhand der Körpergröße und des Gewichts von Menschen, zu treffen.

3.1.3 Unüberwachtes Lernen

Beim unüberwachten Lernen (unsupervised learning) errechnet ein Algorithmus ein statistisches Modell, das die Eingaben anhand derer Attribute unterteilt. Es werden Zusammenhänge bezüglich der Eigenschaften von Daten gefunden. Für die Errechnung des statistischen Modells sind keine gelabelten Trainingsdaten (siehe nachfolgende Kapitel) vonnöten.

Beispiel: k-means clustering

Beim k-means clustering wird eine Punktewolke in k Cluster partitioniert. In Abb. 3.1 wird die Punktewolke in k = 3 Cluster eingeteilt. Die Punkte werden also in drei verschiedene Klassen aufgeteilt.

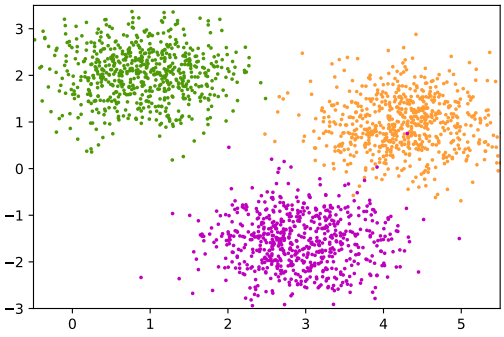

Abbildung 3.1: k-means clustering

3.1.4 Abhängige und unabhängige Variablen

Bei der Prädiktion und Klassifikation werden Eingaben auf Ausgaben abgebildet. Im Machine Learning sind folgende Begrifflichkeiten geläufig:

Unabhängige Variablen
Eingabewerte werden Features, Einflussgröße, vorhersagende Variablen, oder auch unabhängige Variablen genannt.

Abhängige Variablen
Die vorherzusagende Variable, also die Ausgaben des Prädiktionsprogramms, wird abhängige Variable, Zielgröße, Label, Annotation, Target oder Class (Klasse) genannt.

3.1.5 Trainings- und Testdaten

Für das Training und die Evaluation werden Datensätze aus unabhängigen und, im Fall des überwachten Lernens, abhängigen Variablen benötigt. Ein Datensatz (unabhängige Variable 1, unabhängige Variable 2, ..., abhängige Variable 1, abhängige Variable 2, ...) wird Sample genannt. Alle Samples werden in Trainings- und Testdatensätze aufgeteilt. Dies ist wichtig, um mögliches *Overfitting* durch das Testen feststellen zu können. Ein Modell *overfittet* einen Datensatz, wenn es auf diesem gut performt, also gute Vorhersagewerte erzielt, allerdings auf anderen gleichartigen Datensätzen schlecht performt. Das Modell ist zu stark an die Trainingsdaten angepasst und damit nicht generalisierbar. Üblicherweise wird der größere Teil der Datensätze für das Training und ein kleinerer für das Testen verwendet.

3.1.6 Modelle

Die Trainingsphase baut ein statistisches Modell auf, welches zur Vorhersage abhängiger Variablen anhand unabhängiger Variablen verwendet werden kann. Es gibt verschiedene Modelle, die auf unterschiedliche Art und Weise ein statistisches Modell aufbauen.

Beispiel: Für die Vorhersage des Gewichts anhand der Körpergröße (Schreibweise: Gewicht $\sim$ Körpergröße) sind vier Personen mit den (Körpergröße in cm, Gewicht in kg)-Werten (160, 70), (170, 80), (180, 90), (190, 100) bekannt (siehe Abb. 3.2). Im Falle des Modells der linearen Regression wird anhand dieser Werte ein statistisches Modell aufgebaut, die im Graph dargestellte rote Linie (siehe Abb. 3.2). Dieses statistische Modell kann nun verwendet werden, um beliebige weitere Gewichte anhand von Körpergrößen, durch Ablesen des zugehörigen Gewichts anhand der roten Geraden, vorherzusagen. Der Vorgang des Berechnens der abhängigen Variablen durch Anwendung des statistischen Modells auf unabhängige Variablen nennt sich Prädiktion oder Vorhersage.

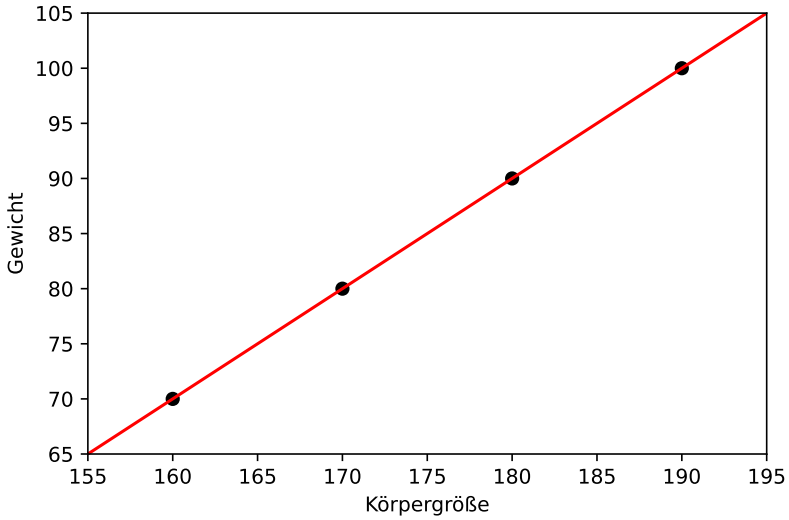

Abbildung 3.2: Lineare Regression

3.1.7 Hyperparameter

Modelle können üblicher Weise durch Hyperparameter parametriert werden. Diese verändern das Modell. Für einige Baum-Modelle kann beispielsweise die maximale Tiefe des Baums angegeben werden. Durch verschiedene Hyperparameter ergeben sich unterschiedliche statistische Modelle durch das Training. Diese unterschiedlichen statistischen Modelle werden in der Regel unterschiedliche Vorhersagen auf den Testdaten erzeugen.

Hyperparameteroptimierung: Rastersuche

Mit dem Ziel möglichst gute statistische Modelle zu erzeugen, sollten geeignete Hyperparameter gewählt werden. Es gibt verschiedene Möglichkeiten geeignete Parameter zu finden. Ein übliches Vorgehen ist die Rastersuche (Grid-Search), bei welcher alle möglichen und einzigartigen Hyperparameter einer vorgegebenen Liste an Hyperparametern für das Training verwendet werden. Die Hyperparameter, die auf dem Testdatensatz die beste Güte erzielen, werden als Hyperparameter gewählt. Für die Güte eines statistischen Modells eignen sich diverse Funktionen, beispielsweise jene, welche in Kapitel 3.1.12 (S. 15) vorgestellt werden.

3.1.8 Ensemble-Methoden

Als Ensemble wird die Kombination mehrerer statistischer Modelle bezeichnet. Ensemble-Methoden verringern die Varianz der Vorhersagen.[1]

[1] https://scikit-learn.org/stable/modules/ensemble.html (Stand: 23.11.2021)

Mittelwertbildungs-Methode

Bei der Mittelwertbildungs-Methode werden mehrere Modelle unabhängig voneinander trainiert und der Mittelwert der Prädiktionen der einzelnen Modelle als Prädiktion des neuen Ensemble-Modells verwendet. Im Fall der Klassifikation wird beispielsweise ein Mehrheitsentscheid getroffen. Ein solches Ensemble-Modell ist der Random-Forest-Klassifizierer, welcher mehrere Baum-basierte Modelle trainiert und benutzt.[2]

Boosting-Methode

Bei der Boosting-Methode werden mehrere Modelle iterativ entwickelt, wobei die nachfolgenden Modelle versuchen den Fehler der vorangehenden Modelle zu reduzieren. Die Prädiktion des Ensemble-Modells ist eine Interpolation der Prädiktionen der Einzelmodelle. Ein solches Ensemble-Modell ist der Gradient-Boosting-Klassifizierer.[3]

3.1.9 Kreuzvalidierung

Eine Form der Kreuzvalidierung ist die *Leave-One-Out-Kreuzvalidierung*. Sie adressiert das Problem, dass, um Overfitting zu vermeiden und/oder zu erkennen, ein Teil der Samples als Testdatensatz verwendet werden muss, wodurch die Menge der Trainingssamples schrumpft. Kreuzvalidierung wird üblicher Weise bei der Hyperparameterwahl angewandt, kann aber generell zum Training und zur Bewertung von Modellen herangezogen werden. Wird die Kreuzvalidierung zur Hyperparameteroptimierung verwendet, so werden die zu testenden Hyperparameter genutzt, um das Modell unter Verwendung der Kreuzvalidierung zu trainieren und jede zu testende Hyperparameterkombination zu evaluieren. Die Hyperparameter, die das beste Modell nach dem Ergebnis der Evaluation, unter Verwendung der Kreuzvalidierung, erzeugt haben, werden als beste Parameter verwendet, um das finale Modell zu trainieren. Anschließend wird dieses Modell mit den Testdaten erneut evaluiert.

Bei der Leave-One-Out-Kreuzvalidierung werden alle Datensätze in k gleich große Datenmengen, den Folds, aufgeteilt (siehe Abb. 3.4, S. 13). In jedem Schritt, dem Split, wird das Modell unter Verwendung von k-1 Folds trainiert. Ein Fold wird zum Testen (Evaluation) verwendet. Das Fold, welches zum Testen verwendet wird, wird in jedem Schritt geändert, sodass kein Fold zweimal zum Testen verwendet wird, bis alle Folds ein mal zum Testen verwendet wurden. Der Durchschnitt der Testergebnisse aller Splits wird als Gesamttestergebnis verwendet. Durch

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (Stand: 23.11.2021)

[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html (Stand: 23.11.2021)

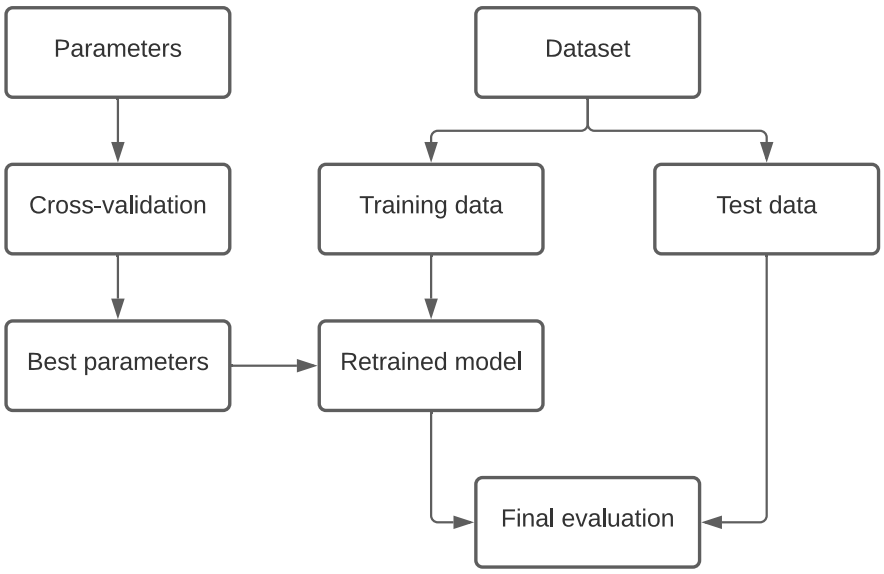

Abbildung 3.3: Ablauf: Hyperparameteroptimierung unter Verwendung der Kreuzvalidierung
[orientiert an https://scikit-learn.org/stable/modules/cross_validation.html]

dieses Vorgehen wurden sowohl alle Datensätze als Trainingsdaten, als auch als Testdaten, verwendet. Wird diese Methode zur Hyperparameteroptimierung verwendet, wird anschließend das mit den besten Hyperparametern trainierte Modell erneut evaluiert. Diese Methode wird auch k-fold Cross-Validation genannt.

3.1.10 Datentransformation

Einige Modelle reagieren empfindlich auf bestimmte Eigenschaften von Rohdaten. Unterschiedliche Wertebereiche und/oder unterschiedliche Verteilungen können zu unterschiedlichen Gewichtungen einzelner Features durch bestimmte Modelle führen. Aus diesem Grund werden Rohdaten häufig normalisiert. Es gibt verschiedene Möglichkeiten Daten zu normalisieren, diese Arbeit beschränkt sich auf die Min-Max-Skalierung und die Standardisierung.

Min-Max-Skalierung

Bei der Min-Max-Skalierung werden alle Werte auf einheitliche Wertebereiche abgebildet, beispielsweise auf den Wertebereich [0, 1].

$$x_{inew} = \frac{x_i - min(x)}{max(x) - min(x)} \tag{3.1}$$

[https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (Stand: 23.11.2021)]

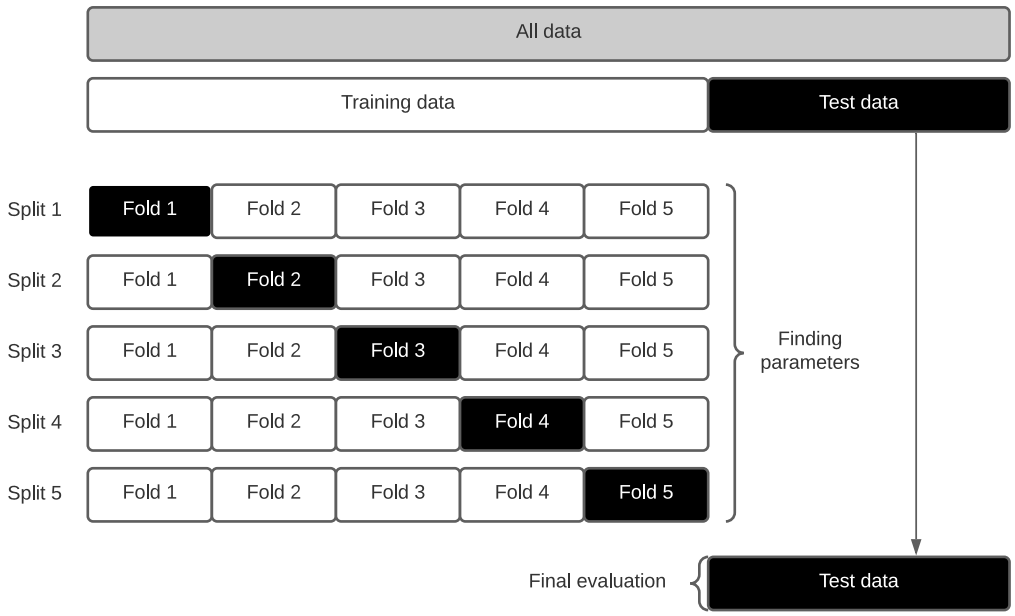

Abbildung 3.4: Hyperparameteroptimierung unter Verwendung der Leave-One-Out-Kreuzvalidierung
[orientiert an https://scikit-learn.org/stable/modules/cross_validation.html (Stand: 23.11.2021)]

Standardisierung

Bei der Standardisierung, oder auch Z-Transformation, werden die Werte so transformiert, dass die transformierten Daten den Mittelwert null und die Standardabweichung eins besitzen.[4]

$$x_{inew} = \frac{x_i - \mu}{\sigma} \tag{3.2}$$

[https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (Stand: 23.11.2021)]

3.1.11 Hauptkomponentenanalyse

Die Hauptkomponentenanalyse (Principal Component Analysis, kurz: PCA) ist eine lineare dimensionale Reduktion unter Verwendung der Singulärwertzerlegung.[5] Sie gehört zu den Algorithmen zur dimensionalen Reduktion der Features (Feature Reduktion, eng: feature reduction) und adressiert damit den „Fluch der Dimensio-

[4] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (Stand: 23.11.2021)

[5] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (Stand: 23.11.2021)

nalität“ (curse of dimensionality). Unter dem Fluch der Dimensionalität versteht man die Probleme, die durch eine hohe Dimension des Feature-Vektors entstehen. Manche Algorithmen benötigen, abhängig von der Dimension der Features, viel Zeit und/oder Platz, wodurch viele Features zu erheblichen Rechenaufwand führen können. Gerade in Kombination mit Hyperparameter-Optimierungs-Verfahren, wie das Grid-Search-Verfahren, kann die Berechnung etlicher Modelle notwendig werden und viele Features damit die Realisierbarkeit solcher Verfahren einschränken. Es gibt verschiedene Algorithmen zur Reduktion der Features. Die PCA rechnet mehrere Dimensionen von Features zu weniger Dimensionen zusammen, wodurch ein Teil der Informationen der Features verloren gehen kann. Die PCA kann parametriert werden durch die Anzahl der beizubehaltenden Komponenten oder durch den Anteil der Varianz der Daten, der mindestens erhalten bleibt. Es besteht die Hoffnung, dass in einem großen Teil der Varianz der Daten, welcher erhalten bleibt, auch ein großer Teil der benötigten Informationen der Daten enthalten ist. Die Reduktion zweier Feature-Dimensionen mittels PCA auf eine Dimension kann als lineare Kombination beider Dimensionen interpretiert werden. Grafisch veranschaulicht wird eine neue Achse in einen zwei dimensionalen Raum eingezeichnet, auf welchen alle Punkte des zwei dimensionalen Raums projiziert werden. Das neue Feature sind die projizierten Punkte auf der neuen Achse. Die Achse wird so eingezeichnet, dass eine maximale Varianz der Daten erhalten bleibt. Die Abbildung 3.7 zeigt einen Datensatz über die Blattbreite und Blattlänge von zwei verschiedenen Pflanzen. Es werden die zwei Hauptkomponenten-Achsen PC-1 und PC-2 eingezeichnet. Nun kann nur die erste Dimension der neuen Punkte der Hauptkomponenten als neues Feature verwendet werden. Im gegebenen Beispiel ist eine Trennung der beiden Pflanzenarten weiterhin möglich.

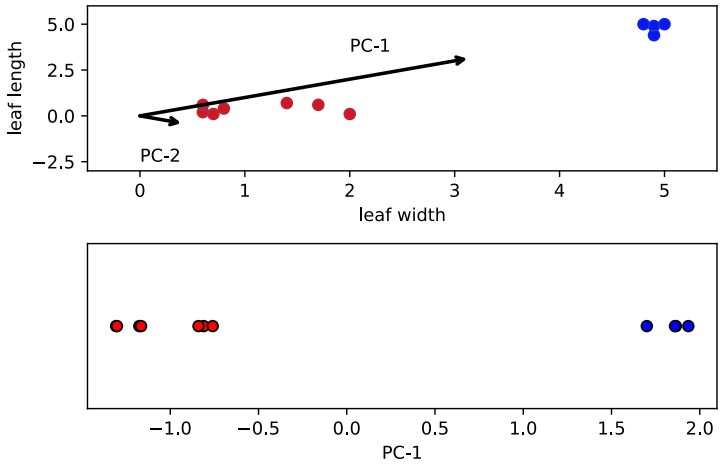

Abbildung 3.5: Principal Component Analysis

3.1.12 Maßzahlen der Modellgüte

Zur Bewertung von Vorhersagemodellen sind diverse Maßzahlen geläufig. Es sei ein Algorithmus zur Prädiktion des Krankheitsstatuses von Patienten mit den Ausprägungen positiv für krank und negativ für nicht krank, also gesund, bekannt. Es handelt sich also um einen binären Klassifikator. Mit diesem Programm wurden insgesamt 11.758 Patienten getestet.

		Tatsächlicher Wert	
		krank	gesund
Testergebnis	positiv	312	120
	negativ	22	11304

Abbildung 3.6: Tabelle zur Klassifikationsgüte

Es ist bekannt, dass von den 11.758 Patienten 11.424 tatsächlich gesund und 334 tatsächlich krank waren. Das Programm klassifiziert von allen untersuchten Personen insgesamt 11.326 als gesund (negativ) und 432 als krank (positiv). Von den 11.424 gesunden Patienten wurden 120 als krank und 11.304 als gesund eingestuft. Von den 334 kranken Patienten wurden 312 als krank und 22 als gesund diagnostiziert. Die eingetragenen Werte der Tabelle Abb. 3.6 entsprechenden den folgenden: Links oben der True-Positive- (TP) Wert, rechts davon der False-Positive- (FP) Wert, in der zweiten Zeile links der False-Negative- (FN) Wert und rechts davon der True-Negative- (TN) Wert. True-Positive steht für einen Patienten der positiv, also krank, klassifiziert wurde und (true) es tatsächlich auch war. Analog sind die übrigen Bezeichnungen zu interpretieren.

Genauigkeit/Accuracy

Die Genauigkeit ist der Anteil der richtig vorhergesagten unter allen vorhersagten Klassen.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

Sensitivität/Recall/Richtig-Positiv-Rate

Die Sensitivität, häufig auch Recall oder Richtig-Positive-Rate (True-Positive-Rate (TPR)) genannt, gibt den richtig klassifizierten Anteil unter Betroffenen an (Fawcett, 2006). Im gegebenen Beispiel also den als krank eingestuften Anteil unter allen Kranken. Sonderfall: Ein Programm, das alle Patienten als krank klassifiziert hat eine Sensitivität von 100%, stellt aber offensichtlich keine guten Klassifikator dar. Aus diesem Grund reicht die alleinige Auswertung dieser Metrik zur Bewertung eines Vorhersagemodells nicht aus.

$$\text{Sensitivität} = \frac{TP}{TP + FN} \tag{3.4}$$

Falsch-Positiv-Rate/Fehlalarm-Rate

Die Falsch-Positiv-Rate (false-positive-rate, FPR) ist der Anteil, der falsch positiv klassifizierten, unter allen nicht betroffenen. Es ist also der Anteil der Negativen, die falsch klassifizierten wurden, unter allen Negativen. (Fawcett, 2006)

$$\text{FPR} = \frac{FP}{FP + TN} \tag{3.5}$$

Positiver Vorhersagewert/Precision

Der positive Vorhersagewert, häufig auch Precision genannt, gibt den richtig klassifizierten Anteil unter positiv getesteten an (Fawcett, 2006). Im gegebenen Beispiel also den Anteil aller Kranken unter allen positiv, also krank, klassifizierten.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.6}$$

F-Score

Der F-Score ist das gewichtete harmonische Mittel der Precision und des Recalls. [6]

$$\text{F-Score} = \frac{2 * precision * recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2} * (FP * FN)} \tag{3.7}$$

Konfusionsmatrix

Eine Konfusionsmatrix (Confusionmatrix) C ist eine Matrix, bei welcher jedes Element $C_{i,j}$ der Anzahl der Samples entspricht, welche der Klasse i zugehörten und für die die Klasse j prädiziert wurden.[7] Die folgende Tabelle zeigt eine Konfusionsmatrix. PC steht für predicted classes, also prädizierte Klassen und AC für actual classes, also tatsächliche Klassen.

PC \ AC	0	1	2
0	10	4	3
1	0	10	0
2	6	0	10

Das Element in der Zeile null und Spalte eins, also die vier, steht für vier Samples, welche zur Klasse eins gehörten, allerdings als Klasse null vorhergesagt wurden.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score (Stand: 23.11.2021)

[7] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html (Stand: 23.11.2021)

Das Element in der Zeile zwei und Spalte null, also die sechs, steht für sechs Samples, welche zur Klasse null gehörten, allerdings als Klasse zwei vorhergesagt wurden. Die Elemente auf der Diagonalen sind die Samples, für welche die Klassen korrekt vorhergesagt wurden.

Anmerkung: Die x- und y-Achse der Konfusionsmatrix werden in unterschiedlichen Quellen vertauscht angegeben. Die hier dargestellte Konfusionsmatrix entspricht der Darstellung des Scikit-Learn-Moduls und wird in dieser Arbeit verwendet.

Makro Durchschnitt

Für jede Klasse der Konfusionsmatrix lassen sich die vorherigen Maßzahlen der Klassifikationsgüte errechnen. Der makro Durchschnitt (macro average, kurz: macro avg) dieser entspricht dem ungewichteten Durchschnitt der Maßzahl der Klassifikationsgüte über alle Klassen. Der Support in der nachfolgenden Tabelle steht für die Anzahl der Samples, welcher einer Klasse zugehörten.

class	precision	recall	support
0	0.87	0.95	2588
1	0.64	0.43	642
2	0.41	0.19	85
3	0.43	0.38	8
4	0.00	0.00	1
macro avg	0.47	0.39	

Der makro Durchschnitt berücksichtigt die Klassenverteilung nicht und wird durch Klassen mit geringem Support verfälscht. Das hier aufgeführt Beispiel besitzt lediglich ein Sample für die Klasse vier, sodass unter Umständen im Trainingsdatensatz kein Sample der Klasse vier vorhanden war und diese Klasse somit nicht trainiert werden konnte. Die Relevanz der Klasse vier könnte im Kontext der Anwendung gering sein, muss jedoch im Einzelfall abgewogen werden. Die *precision* und der *recall* dieser Klasse werden hier dennoch zu einem fünftel in die Gesamtbewertung einbezogen.

Gewichteter Durchschnitt

Für jede Klasse der Konfusionsmatrix lassen sich die vorherigen Maßzahlen der Klassifikationsgüte errechnen. Der gewichtete Durchschnitt (weighted average) dieser entspricht der gewichteten Interpolation über die Anzahl der Samples (support) einer jeden Klasse.

class	precision	recall	support
0	0.87	0.95	2588
1	0.64	0.43	642
2	0.41	0.19	85
3	0.43	0.38	8
4	0.00	0.00	1
weighted avg	0.81	0.33	

Der gewichtete Durchschnitt berücksichtigt die Klassenverteilung. Ist die Klassenverteilung im Testdatensatz nicht relevant, sollen alle Klassen in der Bewertung gleich gewichtet werden, so ist die Verwendung des makro Durchschnitts geeigneter. Sollen die Klassen nach der Häufigkeit ihres Auftretens (im Testdatensatz) bewertet werden, so ist der gewichtete Durchschnitt besser. Auch wenn die Klassen zwei, drei und vier selten vorkommen, könnte es von Interesse sein gerade diese vorherzusagen, z.B. da dies für eine Krankheitsausprägung steht und je höher die Ausprägung, desto wichtiger ist eine Behandlung. Eine höhere Gewichtung der Klassen mit geringerem Support könnte wichtiger sein, als eine Gewichtung nach Support der Klasse.

Durchschnittlicher absoluter Fehler

Sei p_i der prädizierte Wert des i-ten Samples und y_i der korrespondierende echte Wert, dann ist der durchschnittliche absolute Fehler (mean absolute error oder kurz: mean abs. error) über alle $n_{samples}$ Samples:[8]

$$MAE(y, p) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} y_i - p_i \tag{3.8}$$

Grenzwertoptimierungskurve

Die Grenzwertoptimierungskurve, üblicher Weise ROC-Kurve (ROC: Receiver Operating Characteristic) genannt, ist ein Graph in einem Koordinatensystem mit der x-Achse Falsch-Positive-Rate und der y-Achse Richtig-Positiv-Rate zur Beschreibung verschiedener Parametrierungen eines Modells. Die ROC-Kurve wird üblicherweise zur Evaluation binärer Klassifikatoren verwendet. Durch Anpassung des Thresholds, der Klassifikationsschwelle, ergeben sich auf einen Datensatz

[8] https://scikit-learn.org/stable/modules/model_evaluation.html#mean-absolute-error (Stand: 23.11.2021)

unterschiedliche Konfusionsmatrizen und damit unterschiedliche Maßzahlen der Klassifikationsgüte. Eine Diagonale im Koordinatensystem einer binären Klassifikation entspricht einer gleichmäßig, zufälligen Vorhersage der beiden Klassen. Alle Graphen, die über dieser Diagonalen liegen, schneiden besser als der Zufall ab. Je größer die Fläche unter der ROC-Kurve, desto besser. Die maximal mögliche Fläche beträgt eins. Als Vergleichsmetrik wird die Fläche unter dem Graphen (Area Under the Curve, AUC) verwendet. (Fawcett, 2006)

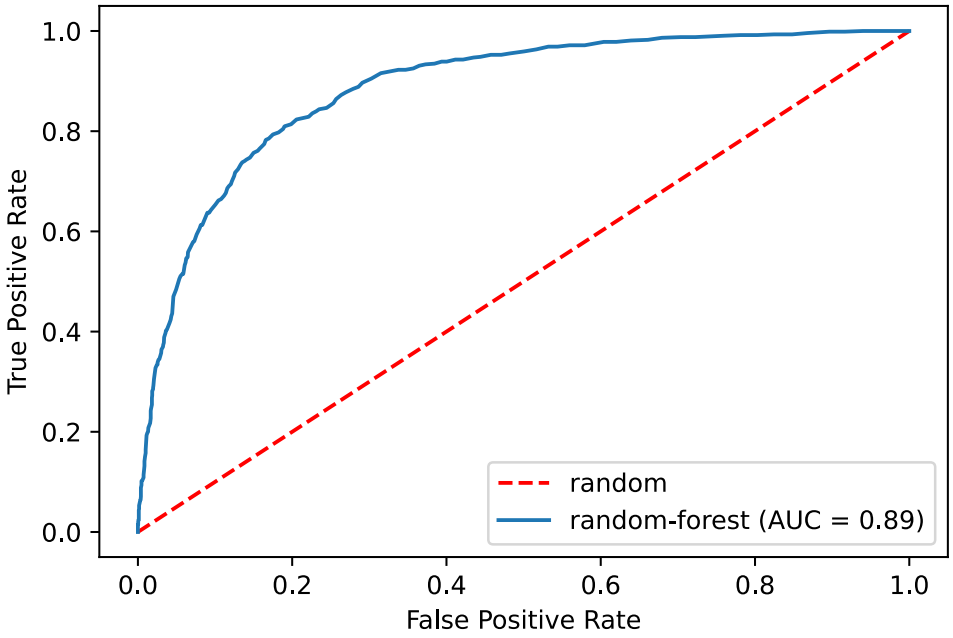

Abbildung 3.7: ROC-Kurve

3.1.13 Dummy-Klassifikatoren

Es gibt verschiedene einfache Strategien zur Erzeugung statistischer Modelle, sogenannte Dummy-Modelle. Üblicherweise wird die Modell-Güte mit der Modell-Güte von Dummy-Modellen verglichen. Im Folgenden eine kurze Erläuterung verwendeter Dummy-Klassifikatoren.

Modell	Beschreibung
stratified	Generiert Vorhersagen anhand der Klassenverteilung der Trainingsdaten
most_frequent	Prädiziert immer die am häufigsten vorkommende Klasse

uniform	Generiert Vorhersagen gleichmäßig zufällig
constant	Prädiziert immer die vorgegebene Klasse

3.1.14 Mustererkennungs-Pipeline

Der Prozess von der Datenermittlung bis hin zur Prädiktion kann mittels einer Pipeline modelliert werden (siehe Abb. 3.8). Im Wesentlichen wird zwischen der Klassifikations- und der Lernphase unterschieden.

Lernphase

In der Lernphase wird ein statistisches Modell mit einer Menge an Daten, dem Lerndatensatz, trainiert. Das Modell passt sich an die Daten an. Der Lerndatensatz sollte möglichst repräsentativ für die Daten der Anwendungsdomäne sein. Ist diese Eigenschaft unzureichend erfüllt, lässt sich das Modell nur eingeschränkt auf die Anwendungsdomäne generalisieren. Trivialerweise kann ein Modell nur Wissen aus Daten extrahieren, welches auch in den Daten enthalten ist. Für diese Arbeitet bedeutet das, dass ein trainiertes Modell nur dann Fehler aus Daten vorhersagen kann, wenn das Wissen über die Fehlerdisposition einer Stelle auch in den gemessenen Softwaremetriken vorhanden beziehungsweise aus den gemessenen Softwaremetriken ableitbar ist.

Die Lernphase wird in vier Schritte, bei der stets die anschließende Phase als Eingabe das Ergebnis der vorangegangenen Phase verarbeitet, unterteilt:

1. **Recording:** Das Sammeln der Rohdaten (R1) und das optionale Sammeln der zugehörigen Klasse (R2), welche für überwachte Lernverfahren notwendig ist. In dieser Phase werden meist mehr Daten erhoben, als im Training benutzt werden.

2. **Preprocessing:** Relevante Daten werden aus den vorangegangen gesammelten Daten gefiltert. Aus den resultierenden Daten sollen die Grundinformationen rekonstruiert werden können, beispielsweise kann eine Audio-Datei noch angehört werden. Eine mögliche Vorverarbeitung wäre das Entfernen des Hintergrunds von Bildern mit Blumen. In dieser Arbeit wird eine Vorselektion zu betrachtender Softwaremetriken, mit dem Ziel des Entfernens ungültiger Softwaremetriken durchgeführt und mit Annotationen kombiniert. Ein Beispiel für eine ungültige Softwaremetrik ist das Messen der Testabdeckung ohne, dass Tests im Quantifizierungsprozess ausgeführt wurden. Da mittels SonarQube bis zu über 70 Metriken gemessen werden können und

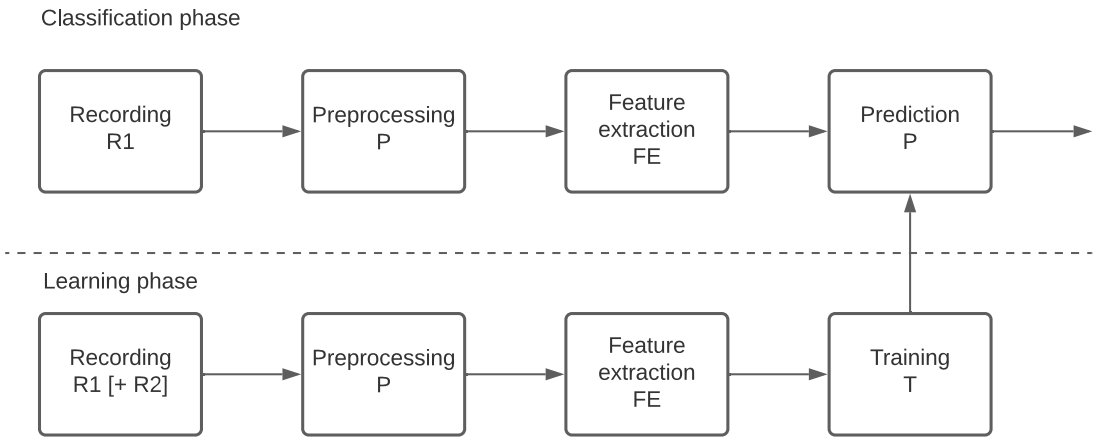

Abbildung 3.8: Mustererkennungs-Pipeline

SonarQube in dieser Arbeit unterstützt wird, ist das Auftreten ungültiger Metriken ein realistisches Szenario und eine Vorselektion eine geeignete Möglichkeit Komplexität in den nachfolgenden Schritten zu reduzieren.

3. **Feature Extraction:** Bei der Extraktion sogenannter Features, werden aus den bestehenden Daten bedeutsame Zahlen gelesen. Es wird gegebenenfalls eine dimensionale Reduktion vorgenommen, also bestimmte Attribute nicht weiter verwendet oder mehrere Attribute zu einer zusammengerechnet. Jedes Attribut, z.B. Blattlänge, wird auch Dimension genannt. Viele Dimensionen enthalten zwar potenziell mehr Informationen, verursachen aber häufig erheblich mehr Rechenaufwand (*Fluch der Dimensionen*).

4. **Training:** Ein statistisches Modell versucht anhand der Features mit möglichst hoher Wahrscheinlichkeit die Annotationen des Lerndatensatzes korrekt vorherzusagen. Im Falle des unüberwachten Lernens wird eine Aufteilung in Klassen ausschließlich durch die Features durchgeführt.

Klassifikations-Phase

In der Klassifikationsphase wird ein bereits trainiertes Modell verwendet, um auf gemessenen Daten die passende Klasse zu bestimmen. In der Nutzung eines Modells werden, auf gleiche Art und Weise, wie in der Lernphase, die Werte der unabhängigen Variablen ermittelt, diese vorverarbeitet (Preprocessing) und Features aus diesen Werten extrahiert. Die extrahierten Features dienen nun dem trainierten Modell zur Bestimmung einer Klasse. In der Klassifikationsphase ist kein Trainingsdatensatz mehr vonnöten. Diese Arbeit beschränkt sich auf die Lernphase.

3.2 Blackboard-Architektur

Der Blackboard-Architektur-Stil enthält drei Komponententypen (siehe Abb. 3.9): Eine Blackboard, Knowledge-Source-Komponenten und Control-Komponenten. Das Blackboard ist ein zentraler Speicher für Informationen, die Knowledge-Source-Komponenten, KS-1 bis KS-n, sind Prozesskomponenten, welche Teilergebnisse liefern. Die Control-Komponenten verarbeiten Teilergebnisse, beobachten das Blackboard und steuern die Knowledge-Source-Komponenten. (Lalanda, 1997) Die Blackboard-Architektur wurde im Kontext der Spracherkennung, welche Parallelen zum Problem dieser Arbeit aufzeigt, manifestiert (Erman et al., 1980).

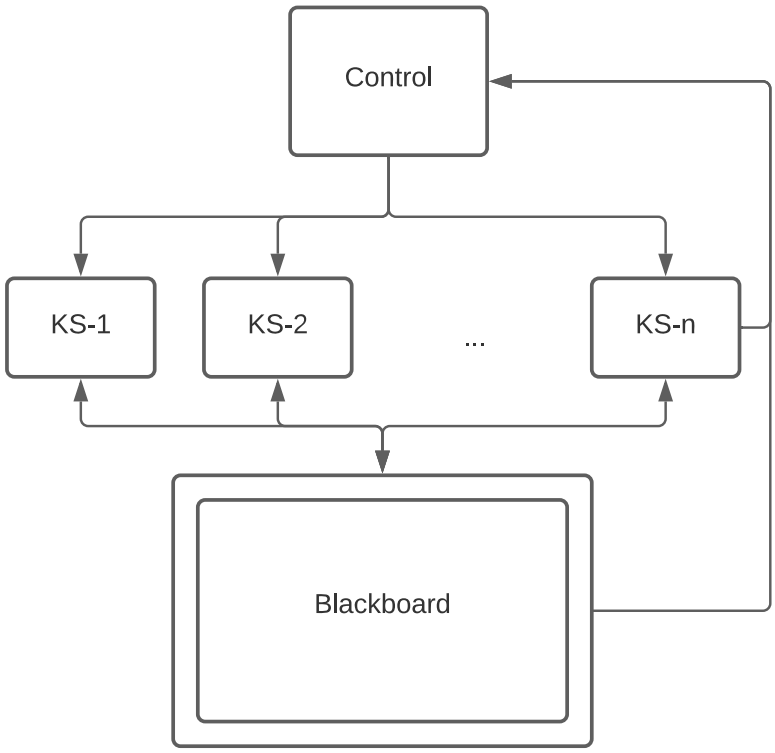

Abbildung 3.9: Blackboard Modell
(Lalanda, 1997, S. 2)

3.3 Softwaremetriken

„[Eine Software-Qualitäts-Metrik ist] eine Funktion, welche Software-Daten [Komponenten] als Eingabe und einen einzelnen numerischen Wert als Ausgabe besitzt, welcher als Grad interpretiert werden kann, in welchem [eine] Software[-Komponente] ein gegebenes Attribut, welches deren Qualität beeinflusst, besitzt.“ („IEEE Standard for a Software Quality Metrics Methodology“, 1994)

3.3.1 Ausgewählte Softwaremetriken

Zur Veranschaulichung des Begriffs der Softwaremetrik werden im folgenden Beispiele aufgeführt und erläutert.

Beispiel 1: Lines Of Code (LOC)

Die Metrik Lines Of Code beschreibt die Anzahl der Programmzeilen, darunter fallen auch Kommentare, üblicherweise aber keine Leerzeilen. Eine verwandte Metrik ist NCLOC, Non-Comment Source Lines, welche die Anzahl der Codezeilen ohne Leerzeilen, Kommentare, Header und Footer, beschreibt.

```
1 // Ein einzeiliger Kommentar
2 function srcFunction(a, b) {
3     if (a > 0) {
4         return a + b;
5     }
6     return 0;
7 }
```

LOC: 7, NLOC: 6

Beispiel 2: Zyklomatische Komplexität

Die zyklomatische Komplexität oder auch McCabe-Metrik ist die Anzahl linear unabhängiger Pfade auf dem Kontrollflussgraphen eines Moduls (McCabe, 1976). Sie gibt die Anzahl möglicher, unterschiedlicher Ausführpfade eines Moduls an. Sowohl die Testbarkeit als auch die Analysierbarkeit werden durch die zyklomatische Komplexität beeinflusst. Für den Kontrollflussgraphen eines Moduls wird sowohl ein expliziter Eingangs-(Entry)-Knoten als auch Ausgangs-(Exit-)Knoten eingeführt (McCabe, 1976).

$$\begin{aligned} &\text{cyclomatic complexity} = e - n + 2 * p \\ &\text{e: Anzahl Kanten im Kontrollflussgraphen} \\ &\text{n: Anzahl Knoten im Kontrollflussgraphen} \\ &\text{p: Anzahl Zusammenhangskomponenten im Kontrollflussgraphen} \end{aligned} \tag{3.9}$$

(McCabe, 1976)

Sei folgender Code gegeben:

```
1 function srcFunction(a, b) {
2     if (a > 0) {
3         return a + b;
4     }
5     return 0;
6 }
```

Der dazugehörige Kontrollflussgraph sieht wie folgt aus; die Codezeile entspricht der Knotennummer.

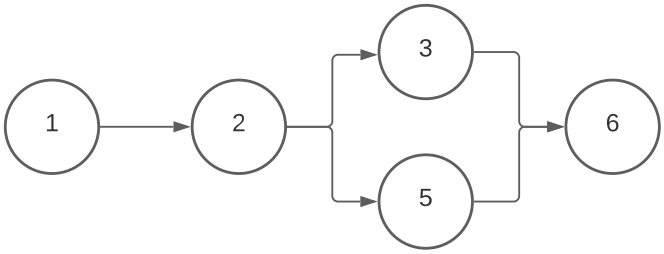

Abbildung 3.10: Kontrollflussgraph

$$\begin{aligned} e &= 5 \\ n &= 5 \\ p &= 1 \\ \text{cyclomatic complexity} &= e - n + 2 * p = 5 - 5 + 2 * 1 = 2 \end{aligned} \tag{3.10}$$

Die möglichen Pfade sind: $1 \to 2 \to 3 \to 6$ und $1 \to 2 \to 5 \to 6$

Beispiel 3: Testabdeckung

Die Testabdeckungs-Metrik gibt an in welchen Grad Quellcode, bei Durchführung von Tests, ausgeführt wird. Es wird zwischen Anweisungs-(Statement-), Verzweigungs-(Branches-), Funktions- und Zeilen-Abdeckung unterschieden. Das Ausführen des nachfolgenden Tests mit der gegebenen Quelldatei ergibt die Testabdeckung, welche in Abb. 3.11 aufgezeigt wird. Eine höhere Testabdeckung reduziert das Risiko unentdeckter Fehler.

```
1 // src.js
2 [...]
3 function srcFunction(a, b) {
4     if (a > 0) {
5         return a + b;
6     }
7     return 0;
8 }
```

```
1 // test.js
2 import {srcFunction} from "src.js";
3
4 describe("Testabdeckung", function() {
5     it("Bei a > 0 wird a und b addiert", function() {
6         expect(srcFunction(1, 2).to.equal(3);
7     });
8 });
```

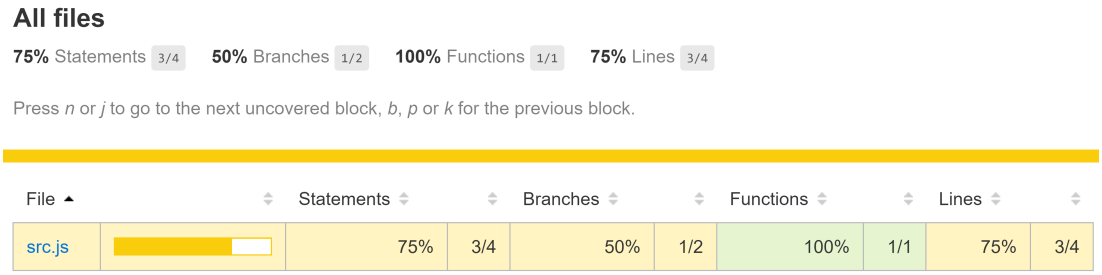

Abbildung 3.11: Testabdeckungsbericht

3.3.2 SonarQube

„SonarQube ist ein automatisches Codereview (Beurteilungs-) Werkzeug zur Detektierung von Fehlern, Schwachstellen und Code Smells in Quellcode. Es kann in einen existierenden Workflow integriert werden, um kontinuierliche Code-Inspektionen über Projekt-Branches und Pull-Requests zu ermöglichen.“[9]

Das Programm SonarQube ist in der Lage Softwaremetriken für 29 verschiedene Programmiersprachen, darunter weit verbreitete wie Java, C#, C, C++ und Python, zu messen („SonarQube“, Stand 23.11.2021). Es ist in der Lage diverse Softwaremetriken, wie beispielsweise Test-Coverage, Cyclomatic-Complexity oder Lines of Code, zu ermitteln. Es bietet ein statisches Code-Analyse-Werkzeug, zur frühzeitigen Identifikation ungünstiger Programmkonstrukte und Hinweise zu Verbesserungen an. Mithilfe von SonarQube kann die Zuverlässigkeit von Programmen, die Applikationssicherheit und die Wartbarkeit verbessert werden („SonarQube“, Stand 23.11.2021). Durch SonarQube ist diese Arbeit auf viele verschiedene Softwareprojekte anwendbar.

Überblick

Die SonarQube-Architektur besteht aus drei Komponenten (siehe Abb. 3.12), dem Analysierer, dem Server und der Datenbank. Beim Analysierer handelt es sich um eine Client-Applikation, einen von mehreren SonarQube Scannern, welcher die Code-Analyse durchführt. Für verschiedene Zielsprachen und Umgebungen werden unterschiedliche Scanner zur Verfügung gestellt. Der SonarQube Scanner analysiert Quellcode, erzeugt daraus einen Snapshot und leitet diesen an den SonarQube Server weiter. Der SonarQube Server verwaltet alle Projekte, welche er in der SonarQube Datenbank ablegt. Über ein Webinterface, bereitgestellt durch den Webserver des SonarQube-Servers können die gemessenen Snapshots vom Endnutzer eingesehen und Konfigurationsänderungen vorgenommen werden.[10]

[9] https://docs.sonarqube.org/9.1/, frei übersetzt (Stand: 15.11.2021)
[10] https://docs.sonarqube.org/9.1/setup/install-server/ (Stand: 15.11.2021)

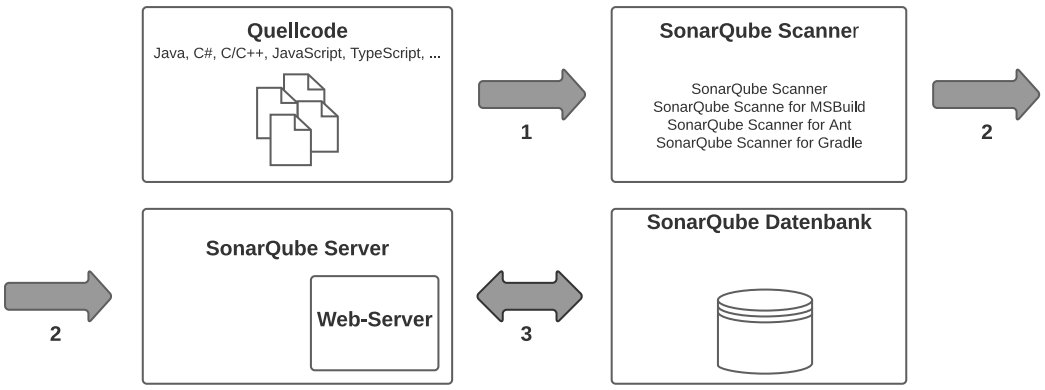

Abbildung 3.12: SonarQube Architektur

Nach absolvierter Analyse und Aktualisierung des SonarQube-Servers stehen die gemessenen Werte im Webinterface zur Verfügung (siehe Abb. 3.13).

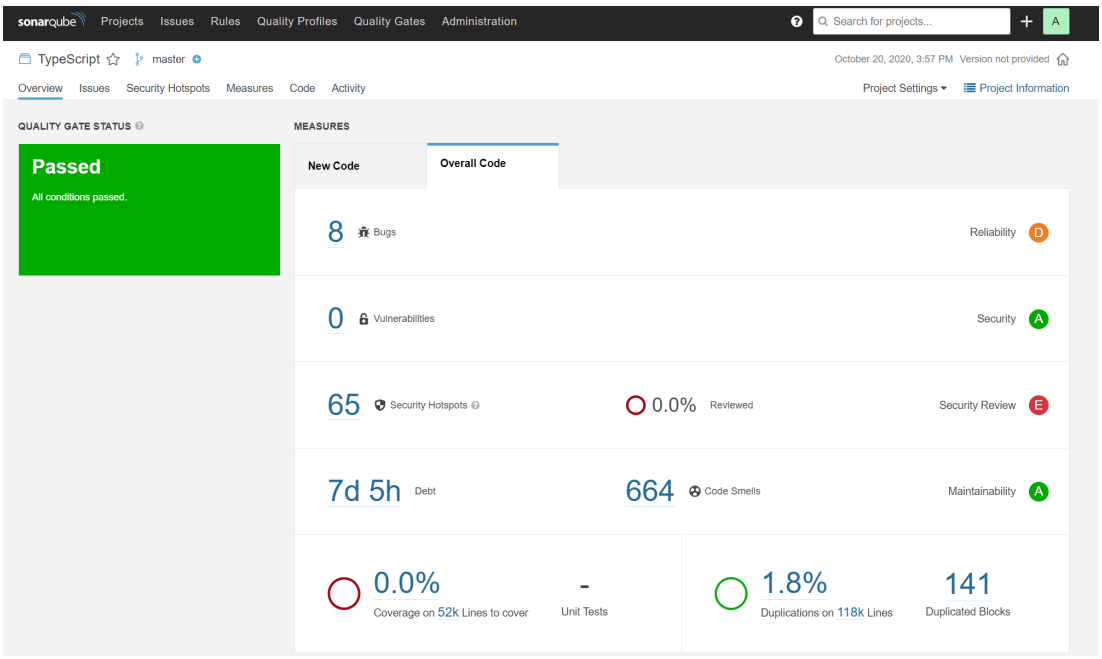

Abbildung 3.13: SonarQube Webinterface

3.4 Repositories und Open Source

Als zu untersuchende Datenquelle dienen dieser Arbeit Open-Source-Repositories, öffentlich zugängliche Datenbanken von Projektartefakten, meist Quellcode. Aufgrund der weiten Verbreitung für Open-Source-Projekte wird die Plattform Github (www.github.com) verwendet.

3.4.1 Open Source

Der Begriff Open-Source wird von der Open-Source-Initiative definiert. Open-Source-Software muss zehn Kriterien erfüllen, fünf von denen lauten wie folgt:

1. **Kostenfreie Redistribution:** Die Lizenz behindert keine Partei vor dem Verkauf oder der Weitergabe der Software als Komponente einer aggregierten Software-Distribution, welche Programme verschiedener Quellen enthält. Die Lizenz sollte keine Lizenzgebühren oder andere Gebühren verlangen.
2. **Quellcode:** Das Programm muss Quellcode enthalten und muss die Distribution in, in Form von Quellcode wie auch in kompilierter Form, erlauben. Wo eine Form eines Produkts nicht mit Quellcode distribuiert wird, muss es eine gut publizierte Möglichkeit geben, Quellcode für nicht mehr als vertretbare Reproduktionskosten, vorzugsweise als kostenloser Download über das Internet, zu erhalten. Der Quellcode muss die bevorzugte Form sein, in welcher Programmierer das Programm modifizieren würden. Bewusst verworrener Quellcode ist nicht erlaubt. Zwischenformen wie die Ausgabe eines Präprozessors oder Übersetzers sind nicht erlaubt.
3. **Abgeleitete Arbeiten:** Die Lizenz muss Modifikationen und abgeleitete Arbeiten erlauben, und muss die Distribution dieser unter der gleichen Lizenz, wie die der Original-Software, erlauben.
4. **Distribution der Lizenz:** Die beigefügten Rechte des Programms müssen für alle gelten, an welche eine Redistribution des Programms durchgeführt wurde, ohne die Notwendigkeit der Ausführung einer zusätzlichen Lizenz dieser Parteien.
5. **Lizenz darf andere Software nicht beschränken:** Die Lizenz darf keine Restriktion auf andere Software einführen, welche mit der lizenzierten Software distribuiert werden. [...]

   [https://opensource.org/osd, frei übersetzt (Stand: 25.11.2021)]

Eine Reihe an Lizenzen werden von der Open-Source-Initiative als Open-Source konform deklariert, darunter die Lizenzen Apache-2.0, BSD-3-Clause, BSD-2-Clause, GPL, LGPL, ISC und MIT. Neben der Open-Source-Initiative gibt es die Free Software Foundation, welche Software als Free Software deklariert, wenn sie ähnliche Anforderungen erfüllt. „Freie Software meint, dass Benutzer die Freiheit besitzen, Software auszuführen, zu editieren, zur Software beizutragen und sie zu teilen.“[11]

[11] https://www.fsf.org/, übersetzt (Stand: 25.11.2021)

3.4.2 Git

„Git ist ein kostenloses und Open Source, verteiltes Versions-Kontroll-System, designt um alles von kleinen bis hin zu sehr großen Projekten schnell und effizient zu handhaben.“ [12] Bekannte Produkte mit ähnlicher Funktionalität sind SVN[13] und Mercurial[14]. Diese Arbeit unterstützt weder SVN noch Mercurial, ist jedoch für diese Systeme erweiterbar gehalten.

Repository

Ein Repository ist eine Datenbank, in der alle Quellcodedateien eines Projekts gehalten werden. Ein leeres Repository wird lokal durch den Befehl `git init` erzeugt, mittels `git clone \$URL` wird ein Repository in ein neues Verzeichnis *geklont*. Der Befehl `git clone https://github.com/microsoft/TypeScript.git` würde das öffentlich zugängliche Repository von TypeScript in das Verzeichnis TypeScript klonen.

Commit

Ein Commit in git zeichnet Änderungen zu einem Repository auf, dies können zum Beispiel neu hinzugefügte Dateien oder Änderungen in Dateien sein. Jeder Commit erhält eine eindeutige ID, die *Commit hash*. Durch diese ID kann der exakte Zustand eines Commits, also des Projektes, wiederhergestellt werden. Des Weiteren werden Metadaten, wie der Autor-Name oder die Autor-E-Mail, aber auch die Eltern-Commit-hashes zu jedem Commit gespeichert. Wird ein Repository geklont, so sind alle vorangegangen Projektzustände mittels git rekonstruierbar. Diese Projektzustände werden in dieser Arbeit untersucht. Sie werden auf Fehlerbehaftung analysiert (Annotation) und Softwaremetriken der Quelldateien gemessen (Quantifizierung).

3.4.3 Github

Github ist eine Platform, die über 200 Millionen Repositories bereitstellt und eine Community von über 65 Millionen Menschen besitzt[15]. Diese Plattform, auf welche mit git zugegriffen werden kann, besitzt eine Vielfalt von Open-Source-Projekten, unter anderem auch von namhaften, großen Unternehmen wie Google und Microsoft. Github als Plattform dient dieser Arbeit sowohl als Quelle für zu untersuchende Projekte, als auch als Plattform zur Veröffentlichung der Ergebnisse.

[12] https://git-scm.com/, übersetzt (Stand: 22.11.2021)
[13] https://subversion.apache.org/ (Stand: 23.11.2021)
[14] https://www.mercurial-scm.org/ (Stand: 23.11.2021)
[15] https://github.com/about (Stand: 28.06.2021)

3.5 Arc42

Arc42 ist ein Template zur Aufsammlung architekturrelevanter Informationen (Hruschka & Starke, (Stand: 23.11.2021)). Es umfasst zwölf Kapitel, auch Schubladen bezeichnet. In der Schubladen-Analogie sollen alle Artefakte zur Beschreibung einer Softwarearchitektur in die jeweils passende Schublade gesteckt werden. Dieses Template gibt an was und wie dokumentiert werden soll. Die Umsetzung dieser Arbeit wird Arc42-konform dokumentiert.

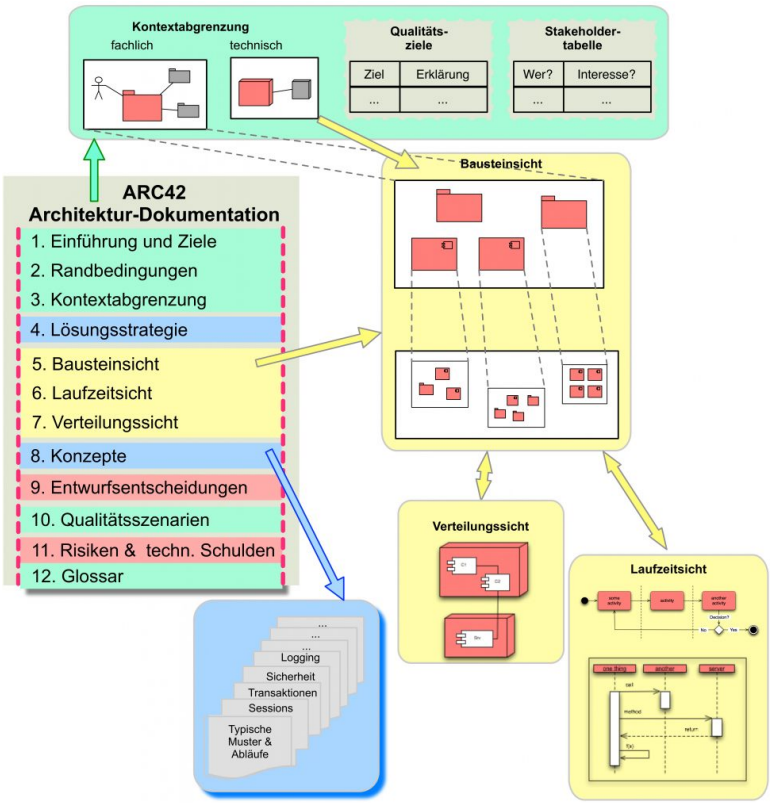

Abbildung 3.14: arc42 Überblick
(Hruschka & Starke, (Stand: 23.11.2021))

3.6 Anforderungsspezifikation

Die Anforderungsspezifikation ist Teil der Architekturbeschreibung nach dem Arc42-Template. Anforderungen werden üblicherweise in funktionale und qualitative Anforderungen unterteilt.

3.6.1 Funktionale Anforderungen

Funktionale Anforderungen können auf unterschiedliche Art und Weise spezifiziert werden. In dieser Arbeit wird die Struktur nach Mavin et al., 2009, verwendet. Sie adressiert Doppeldeutigkeit, Vagheit, Komplexität, Auslassungen, Wiederholungen, Weitschweifigkeit, unangemessene Umsetzung (Anmerkungen, welche eher die Umsetzungsstrategie beschreiben, als die Anforderungen) und Untestbarkeit (Mavin et al., 2009, z.T. übersetzt).

Es wird die generische Syntax `<optional preconditions> <optional trigger> the <system name> shall <system response>` eingeführt.
Vorbedingungen (Preconditions) Bedingungen, bei welcher die Anforderung umgesetzt werden kann. **Auslöser (Trigger)** Das Ereignis, welches die Anforderung initiiert. **Systemname (System name)** Name des Systems, welches die Anforderung umsetzt. **Systemantwort (System response)** Das notwendige Systemverhalten.

(Mavin et al., 2009)

Jede Anforderung wird nach dieser Syntax formuliert. In dieser Arbeit wird die tabellarische Interpretation dieser Schreibweise verwendet. Es werden für unterschiedliche Klassen an Anforderungen eigene Schlüsselwörter vorgeschlagen und die typische Syntax aus der generischen Form abgeleitet:

Allgegenwärtige Anforderungen

```
The <system name> shall <system response>
```

(Mavin et al., 2009)

Ereignisgesteuerte Anforderungen

```
WHEN <optional preconditions> <trigger> the <system name>
    shall <system response>
```

(Mavin et al., 2009)

Unerwünschtes Verhalten

```
IF <optional preconditions> <trigger>, THEN the <system name>
    shall <system response>
```

(Mavin et al., 2009)

3.6.2 Qualitative Anforderungen

„Unter Softwarequalität versteht man die Gesamtheit der Merkmale und Merkmalswerte eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen.“ (Balzert, 1998, S.257)

Produkt-Qualitäts-Modell ISO/IEC 25010

Häufige Verwendung finden die Begriffsmodelle nach ISO/IEC 9126 bzw. 25010, welche erstere ablöst. Acht Kategorien an Softwarequalitäten werden in dieser Norm aufgeführt, welche durch jeweilige Subcharakteristiken aufgebaut werden.

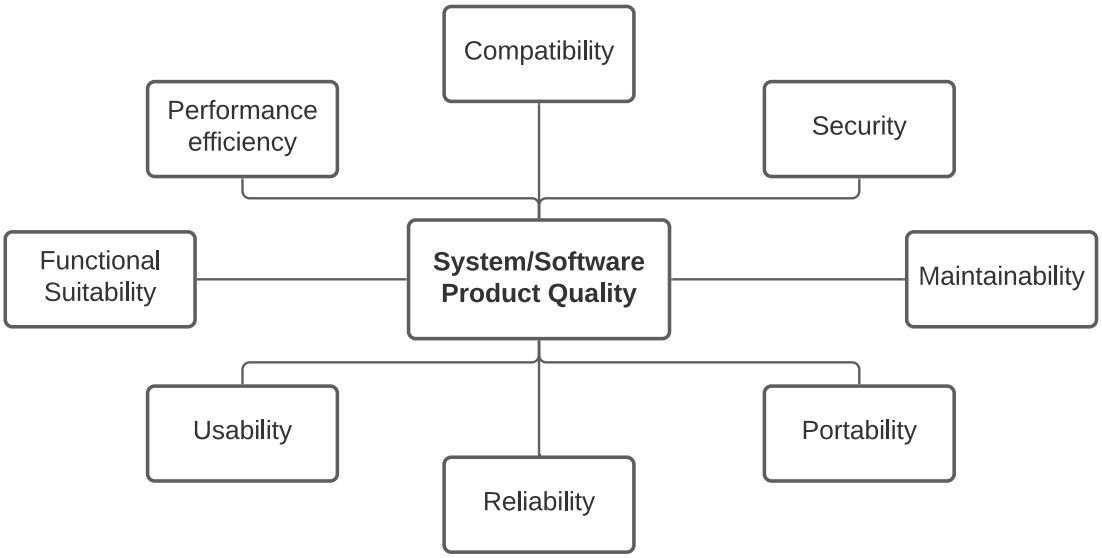

Abbildung 3.15: Produkt-Qualitäts-Modell („Norm ISO/IEC 25010“, 2011)

Die Qualitätsdimensionen unterteilen sich in folgende Subcharakteristiken:

Funktionale Eignung: Funktionale Vollständigkeit, funktionale Korrektheit, funktionale Angemessenheit

Performanz-Effizienz: Zeitverhalten, Ressourcennutzung, Kapazität

Sicherheit: Vertraulichkeit, Integrität, Nicht-Abstreitbarkeit, Rechenschaftspflicht, Authentizität

Kompatibilität: Koexistenz, Interoperabilität

Wartbarkeit: Modularität, Wiederverwendbarkeit, Analysierbarkeit, Modifizierbarkeit, Testbarkeit

Nutzbarkeit: Angemessenheit, Wiedererkennbarkeit, Erlernbarkeit, Operabilität, Benutzerfehlerschutz, Benutzerschnittstellenästhetik, Zugänglichkeit

Zuverlässigkeit: Reife, Verfügbarkeit, Fehlertoleranz, Wiederherstellbarkeit

Portabilität: Anpassungsfähigkeit, Installierbarkeit, Austauschbarkeit

Anforderungsspezifikation durch Qualitätsszenarien

Eine qualitative Anforderung kann durch ein Qualitätsszenario, einer „[...] Qualitäts-Attribut-spezifische[n] Anforderung“ (Bass et al., 2003, S. 91, übersetzt), angegeben werden (Bass et al., 2003). Ein solches Szenario besteht aus sechs Teilen:

Ursprung des Stimulus (Source of stimulus)
Eine Entität, z.B. ein Mensch, ein Computer, System, oder ein beliebiger weiterer Aktuator, welcher den Stimulus generiert hat.
Stimulus (Stimulus)
Der Stimulus ist eine Bedingung, welche berücksichtigt werden muss, wenn sie am System ankommt.
Umgebung (Environment)
Der Stimulus tritt unter bestimmten Bedingungen auf. Das System könnte in einer Overload-Bedingung sein oder könnte laufen, wenn der Stimulus eintritt, oder eine andere Bedingung könnte wahr sein.
Artefakt (Artifact)
Ein Artefakt wird stimuliert. Dies kann das ganze System oder Teile des Systems sein.
Antwort (Response)
Die Antwort ist die Aktivität, welche durchgeführt wird, nachdem der Stimulus aufgetreten ist.
Antwort-Messung (Response measure)
Die auftretende Antwort sollte in einer Art messbar sein, dass die Anforderung getestet werden kann.

(Bass et al., 2003, S. 91, z.T. übersetzt)

3.7 Implementierung

Dieses Kapitel beschreibt die verwendeten Programmiersprachen, Programme und Bibliotheken der Implementierung.

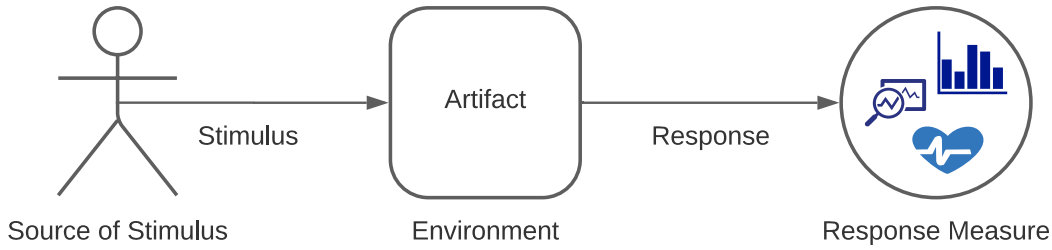

Abbildung 3.16: Qualitätsszenario
(orientiert an: Bass et al., 2003, S. 92)

3.7.1 Node.js

Node.js ist eine plattformübergreifende, asynchrone, ereignisgesteuerte Laufzeitumgebung für JavaScript. Sie „ist designt worden, um skalierende Netzwerkapplikationen zu entwickeln“[16] und wurde auf Basis der Chrome V8 JavaScript engine, „Googles open source hochperformante JavaScript und WebAssembly“[17] Implementierung, entwickelt.

JavaScript ist eine Skriptsprache, die ursprünglich für die Nutzung in Webbrowsern verwendet wurde. Durch die Entwicklung von Node.js erweitert sich das Anwendungsspektrum dieser Sprache. Durch die plattformübergreifende Anwendbarkeit der Sprache JavaScript ergeben sich weitere Anwendungsszenarien für Entwickler, wie die Wiederverwendbarkeit bestimmter Komponenten für Applikationen mit unterschiedlichen Auslieferungs-Umgebungen, wie Android, iOS oder Desktop-Plattformen. Im Browser sind bestimmte Funktionalitäten, welche *General-purpose languages* (Allzwecksprachen) üblicherweise unterstützen, nicht gewünscht. Beispielsweise ist in einer Browser-Umgebung der Zugriff auf das Dateisystem des Host-Rechners aus Sicherheitsgründen nicht uneingeschränkt möglich. Node.js bietet zur Schließung dieser Lücke eine eigene API an.

3.7.2 Npm

Npm (ehemals node package manager) ist eine Webseite, ein Command Line Interface (Kommandozeileninterpreter, CLI) und die weltweit größte Software-Registry (Software-Registrierungsdatenbank).[18] Mittels npm können Pakete erzeugt, geteilt, heruntergeladen und installiert werden.

[16] https://nodejs.org/en/, übersetzt (Stand: 23.11.2021)
[17] https://v8.dev/, übersetzt (Stand: 23.11.2021)
[18] https://docs.npmjs.com/about-npm (Stand: 23.11.2021)

Webseite

Die npm-Webseite ermöglicht das Durchsuchen von veröffentlichten Paketen der npm-Registry. Auf der Paket-spezifischen Seite befinden sich Meta-Daten wie die wöchentliche Download-Zahl des Pakets, die Version, die Lizenz und gegebenenfalls eine Homepage oder ein öffentliches Repository, häufig eins aus Github. Meistens ist eine kurze Beschreibung und Einführung zur Nutzung des Pakets vorhanden (siehe Abb. 3.17).

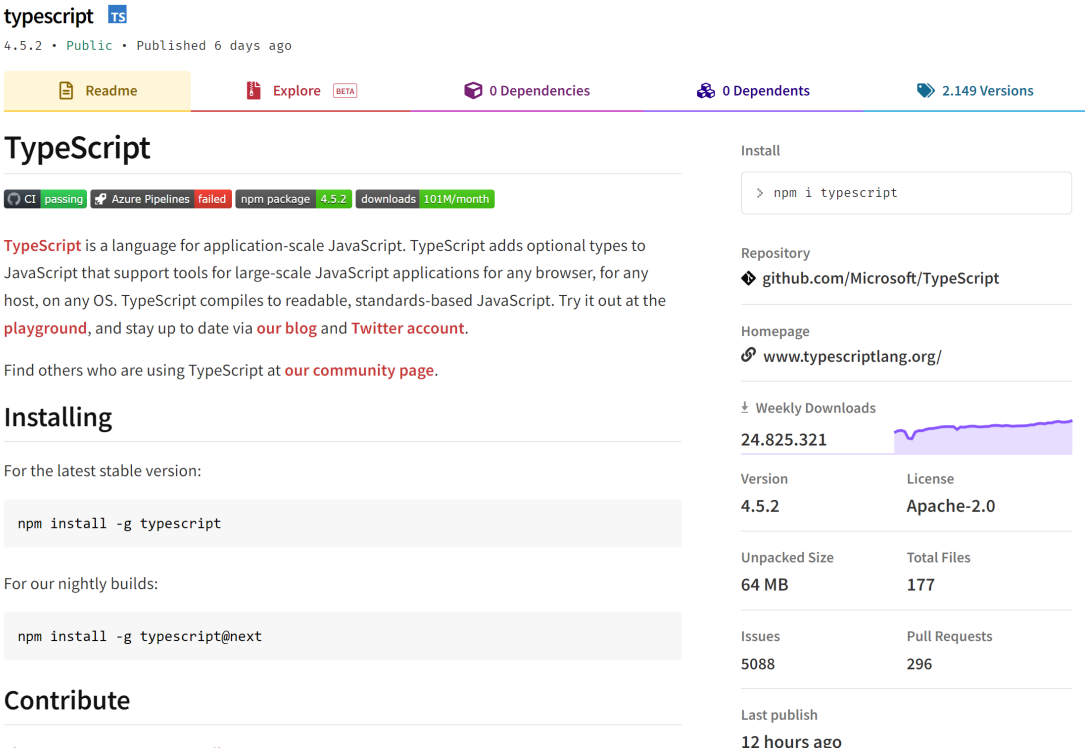

Abbildung 3.17: Npm Webseite: TypeScript [https://www.npmjs.com/package/typescript]

3.7.3 TypeScript

TypeScript ist eine objektorientierte Open-Source-Sprache und ein Superset der Sprache JavaScript. Diese Sprache zeichnet sich insbesondere durch ihre plattformübergreifende Anwendbarkeit, Typisierung und der einfachen Einbindung vieler Open-Source-Bibliotheken mittels npm aus.[19] [20] [21]

[19] https://github.com/microsoft/TypeScript (Stand: 23.11.2021)
[20] https://www.npmjs.com/package/typescript (Stand: 23.11.2021)
[21] https://www.typescriptlang.org/ (Stand: 23.11.2021)

3.7.4 Dependency Injection

Dependency Injection (Abhängigkeitsinjektion) ist ein Entwurfsmuster, bei welchem Abhängigkeiten eines Objekts zur Laufzeit bestimmt und gesetzt werden. Die Bezeichnung *Dependency Injection* wurde vom Softwareentwickler Martin Fowler eingeführt, um das Konzept, welches unter dem generischen Namen *Inversion of Control* bekannt war, zu präzisieren. (Fowler, 2004) Der Grundgedanke der Dependency Injection (DI) ist es, die Verantwortlichkeit der Abhängigkeiten zwischen den Objekten eines Programms (also den Klassen und ihren Abhängigkeiten) und deren Instanziierung in einer zentralen Komponente anzugeben. Da sich dieses Dokument auch als Dokumentation der Software versteht und für das Verständnis der Verwendung einzelner Plugins Kenntnisse über DI mit InversifyJS benötigt werden, finden Sie eine kurze Einführung zu InversifyJS in Anhang A. Diese richtet sich primär an Entwickler, welche diese Arbeit nutzen wollen.

Dependency Inversion Principle (Abhängigkeits-Umkehrungs-Prinzip)

Unter dem Dependency Inversion Principle versteht man eine spezielle Art der Entkopplung zweier Softwaremodule durch Abstraktionen. Seien A und B zwei Klassen und A referenziert die Klasse B (siehe Abb. 3.18). Durch Einführung einer Schnittstelle dependendInterface, ist es möglich die Klasse A von der Klasse B zu entkoppeln. A referenziert nun eine Schnittstelle, die konkrete Implementierung dieser ist entkoppelt. Durch Übergabe einer beliebigen Realisierung der Schnittstelle dependendInterface an die Klasse A, kann die Instanz der Realisierung beliebig und dynamisch ausgetauscht werden. A erzeugt nun nicht mehr selbst seine Abhängigkeit (B), sondern kriegt sie von einer aufrufenden Klasse respektive einem Dependency Injection Framework übergeben. (Robert C., 2003)

Die Übergabe einer Abhängigkeit, z.B. über den Konstruktor oder eine Setter-Methode, ist gängige Praxis in Dependency-Injection-Frameworks. Durch das Arbeiten auf Abstraktionen ergibt sich eine besonders einfache Austauschbarkeit bestehender Implementierungen.

InversifyJS

„InversifyJS ist ein leichtgewichtiger Inversion of Control (IoC) Behälter für TypeScript- und JavaScript-Applikationen. Ein IoC-Behälter nutzt Klassenkonstruktoren zur Identifizierung und Injektion derer Abhängigkeiten. “ (Jansen, 2021, frei übersetzt)

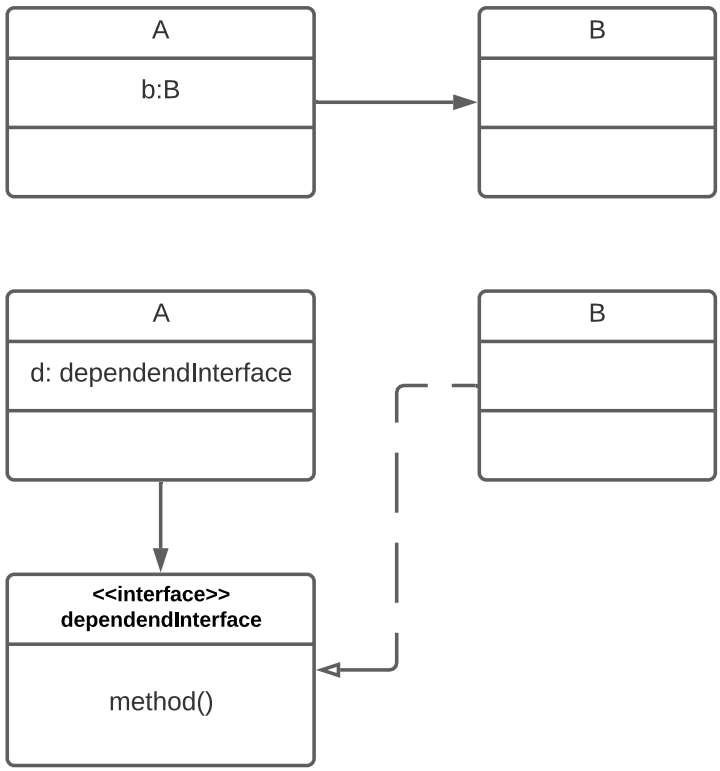

Abbildung 3.18: Dependency Inversion (Abhängigkeitsumkehrung)

3.7.5 MongoDB

„MongoDB ist eine dokumentenorientierte Datenbank, die auf einfache Entwicklung und Skalierung ausgelegt ist.“ („MongoDB Dokumentation“, eingesehen am 15.11.2021, frei übersetzt) Mit wöchentlichen Downloadzahlen des MongoDB-Treibers für Node.js von über 3.000.000[22] besitzt MongoDB eine hohe Popularität.[23]

MongoDB teilt seine Datenbank in Collections (Kollektionen) auf, welche wiederum Dokumente enthalten. Dokumente enthalten Feld-Werte-Paare, wie sie aus dem JSON-Format bekannt sind. Dokumente einer Kollektion können unterschiedliche Attribute besitzen, wie in Abb. 3.19 zu sehen ist: Zwei Dokumente der Kollektion *Atmungssysteme* besitzen zum Teil unterschiedliche Attribute. Ein Dokument enthält lediglich einen *Namen*, ein weiteres Dokument einen *Namen* und ein Attribut *icdTen*.

[22]https://www.npmjs.com/package/mongodb, (Stand: 22.11.2021)
[23]https://www.mongodb.com/de (Stand: 23.11.2021)

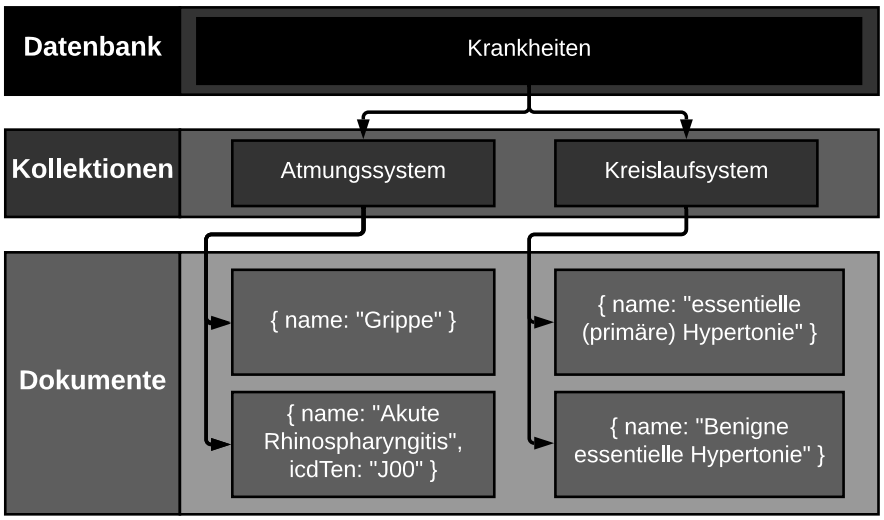

Abbildung 3.19: MongoDB Aufbau

MongoDB Compass

Das Programm MongoDB Compass (siehe Abb. 3.20, S. 38) ist eine GUI für MongoDB, mit welcher visuell Daten einer MongoDB erkundet werden können. Es bietet volle Unterstützung für CRUD-Operationen, also Create-Read-Update-Delete-Operationen (Erzeugen, Lesen, Aktualisieren und Entfernen).[24]

3.7.6 Python

„Python ist ein Interpreter und eine Objekt-orientierte, höhere Programmiersprache [...]“[25]. Sie findet häufig Anwendung im Bereich des maschinellen Lernens.

3.7.7 Jupyter Notebook

Jupyter Notebook ist „ein Veröffentlichtungsformat für reproduzierbare computergestütze Arbeitsabläufe“ (Kluyver et al., 2016, übersetzt). Es ermöglicht das dynamische Ausführen einzelner Quellcode-Zellen unter Verwendung des letzten Programmzustandes. Dies ermöglicht eine zügige, iterative Entwicklung und unterstützt dadurch experimentelle Verfahren, wie sie im maschinellen Lernen geläufig sind. Des Weiteren können Mark-Down-Zellen erzeugt werden, welche den Quellcode durch Dokumentation ergänzen (siehe Abb. 3.21, S. 39).

[24] https://www.mongodb.com/products/compass (Stand: 22.11.2021)
[25] https://www.python.org/doc/essays/blurb/ (Stand: 23.11.2021)

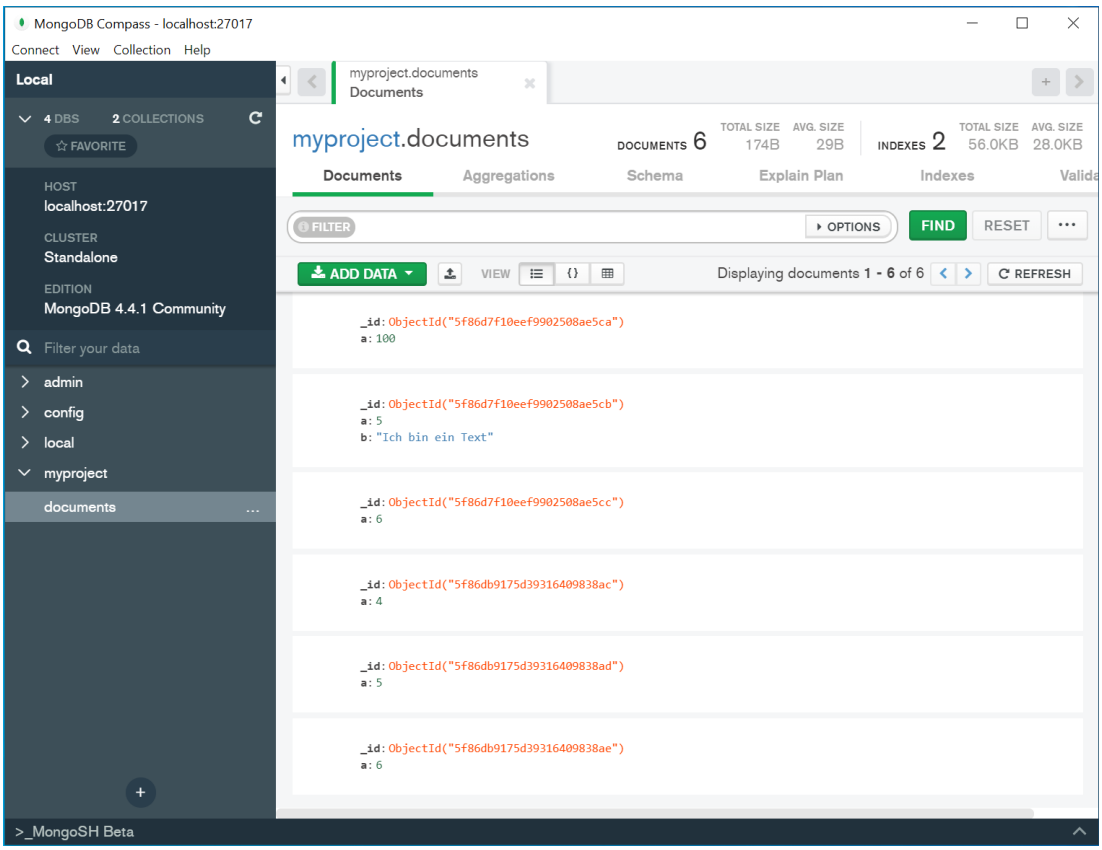

Abbildung 3.20: MongoDB Compass

3.7.8 Anaconda

Anaconda ist ein data science toolkit (Werkzeugkoffer für die Datenwissenschaften), welches ermöglicht, mit mehreren tausend Open-Source-Paketen und Bibliotheken, zu arbeiten. Es unterstützt das Management von Python-Umgebungen inklusiver der Wahl der Python-Version und wird von über 25 Millionen Nutzern weltweit verwendet.[26]

3.7.9 Scikit-Learn

„Scikit-learn ist ein Python-Modul, das eine breite Palette modernster Algorithmen des maschinellen Lernens, für mittelgroße überwachte und unüberwachte Probleme, integriert.“ (Pedregosa et al., 2011, übersetzt) Das Modul überzeugt durch eine verständliche Dokumentation, gute Schnittstellen-Ästhetik, einfache Erlernbarkeit und durch Zuverlässigkeit. Unterstützt werden unter anderem Algorithmen für die Klassifikation, Regression, für das Clustering, die Datenvorverarbeitung, Modell-Selektion und die dimensionale Reduktion. Scikit-Learn wird unter einer BSD-

[26] https://www.anaconda.com/products/individual (Stand: 23.11.2021)

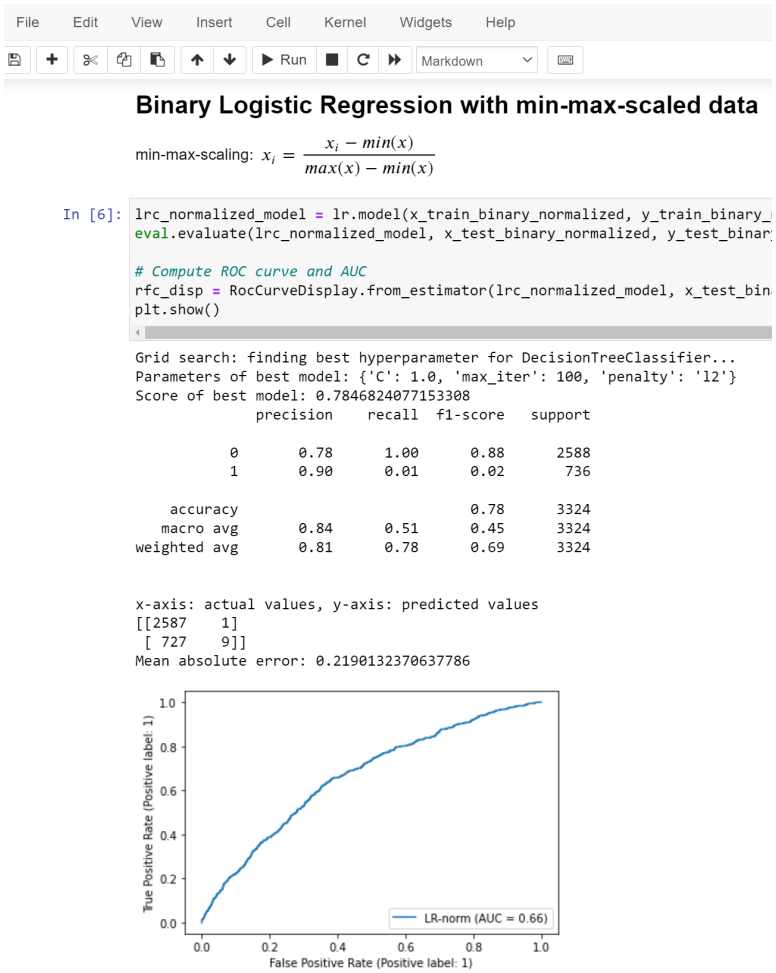

Abbildung 3.21: Jupyter Notebook

Lizenz, einer freizügigen Open-Source-Lizenz, veröffentlicht und ist damit auch für die kommerzielle Nutzung frei verfügbar.[27]

[27] https://scikit-learn.org/stable/ (Stand: 23.11.2021)

4 Umsetzung

Im Folgenden werden, orientiert am arc42-template, softwarerelevante Architekturartefakte aufgeführt.

4.1 Einführung und Ziele

Ziel ist die Entwicklung eines erweiterbaren und veränderbaren Programms zur Quantifizierung von Softwareprojekten und Prädiktion von Fehlerstellen anhand der gemessenen Werte einer Stelle. Dieses Programm soll weiteren Untersuchungen zur Programm-Fehler-Ursachen-Findung dienen und kann infolgedessen zur kommerziellen Nutzung weiterentwickelt werden. Im Rahmen der Evaluierung der Softwarearchitektur soll eine konkrete Implementierung Zusammenhänge zwischen Softwaremetriken und Fehlerauftreten feststellen. Im Rahmen dieser Arbeit soll sowohl ein Framework, als auch eine konkrete Implementierung unter Verwendung des Frameworks durchgeführt werden. Es gibt sowohl generische Anforderungen an das Framework, also auch konkrete Anforderungen an die Implementierung. Zu jeder generischen Anforderung wurde mindestens eine konkrete Anforderung gestellt.

4.1.1 Anforderungsüberblick: Framework

In diesem Projekt werden überwachte Lernverfahren, welche zunächst eine Menge von Tupeln an Ein- und Ausgabewerten benötigen, betrachtet. Beispielsweise kann für ein Projekt für jede Datei die Anzahl an Zeilen an Code gemessen werden und für jede Datei angegeben werden, ob sie einen Fehler enthält oder nicht. Diese Menge an Daten dient dann dem Modelltraining. Das resultierende Modell kann nun genutzt werden, um aus reinen Messwerten, z.B. der Messung der Anzahl an Zeilen einer konkreten Datei, zu prädizieren, ob sich in dieser Datei ein Fehler befindet oder nicht. Das Framework bietet Komponenten, welche diese abstrakten Anforderungen unterstützen. Die Anforderungen an das Framework werden im Folgenden aufgelistet.

1. Erzeugung eines Lerndatensatzes

Vorbedingungen	Eine Datenbank mit Lokalitäten, Annotationen und Quantifizierungen ist gegeben.
Systemname	Preprocessing
Systemverhalten	Es wird eine Menge aus Tupeln (Annotation, Quantifizierungen) erzeugt. Die Tupel ordnen jeder Lokalität einer Annotation und Quantifizierungen zu. Die Annotation gibt die Fehlerbehaftung der Lokalität an. Die Quantifizierungen enthalten metrische Werte, welche aus der Lokalität abgeleitet werden können.

1.1 Messen von Lokalitäten

Vorbedingungen	Ein zu untersuchendes Projekt mit Lokalitäten ist gegeben.
Trigger	Das Programm zum Messen der Lokalitäten wird gestartet.
Systemname	LocalityRecording
Systemverhalten	Lokalitäts-Objekte werden erzeugt.

1.2 Persistieren von Lokalitäten

Vorbedingungen	Lokalitäts-Objekte sind gegeben.
Trigger	Die Lokalitäten wurden gemessen.
Systemname	LocalityRecording
Systemverhalten	Lokalitäts-Objekte werden in einer Datenbank persistiert.

1.3 Vorverarbeiten von Lokalitäten

Vorbedingungen	Lokalitäts-Objekte sind gegeben.

Trigger	Das Programm zum Vorverarbeiten von Lokalitäten wird gestartet.
Systemname	LocalityPreprocessing
Systemverhalten	Eine Teilmenge der Lokalitäts-Objekte wird gefiltert und anschließend in einer Datenbank persistiert.

1.4 Messen von Quantifizierungen

Vorbedingungen	Lokalitäts-Objekte sind gegeben. Das Projekt, aus welchem die Lokalitäts-Objekte gemessen wurden, ist gegeben.
Trigger	Das Programm zum Messen der Quantifizierungen wird gestartet.
Systemname	Quantifying
Systemverhalten	Es wird ein Quantifizierungs-Objekt zu jedem Lokalitäts-Objekt erzeugt, sodass Tupel aus Lokalitäts- und Quantifizierungs-Objekten entstehen. Quantifizierungs-Objekte enthalten Zahlenwerte, welche aus der Lokalität abgeleitet werden.

1.5 Persistieren von Quantifizierungen

Vorbedingungen	Es sind Quantifizierungs-Objekte gegeben.
Trigger	Die Quantifizierungen wurden gemessen.
Systemname	Quantifying
Systemverhalten	Tupel aus Lokalitäts- und Quantifizierungs-Objekten werden in einer Datenbank persistiert.

1.6 Erzeugung von Annotationen

Vorbedingungen	Lokalitäts-Objekte sind gegeben. Das Projekt, aus welchem die Lokalitäts-Objekte gemessen wurden, ist gegeben.

Trigger	Das Programm zum Erzeugen von Annotationen wird gestartet.
Systemname	Annotating
Systemverhalten	Es wird ein Annotations-Objekt zu jedem Lokalitäts-Objekt erzeugt, sodass Tupel aus Lokalitäts- und Quantifizierungs-Objekten entstehen. Annotations-Objekte enthalten einen Wert zur Angabe der Fehlerbehaftung einer Lokalität.

1.7 Persistieren von Annotationen

Vorbedingungen	Es sind Annotations-Objekte gegeben.
Trigger	Die Annotationen wurden erzeugt.
Systemname	Annotating
Systemverhalten	Tupel aus Lokalitäts- und Annotations-Objekten werden in einer Datenbank persistiert.

1.8 Erzeugen eines Lerndatensatzes

Vorbedingungen	Es sind Tupel aus Lokalitäts- und Quantifizierungs-Objekten gegeben. Es sind Tupel aus Lokalitäts- und Annotations-Objekten gegeben.
Trigger	Das Programm zum Erzeugen eines Lerndatensatzes wird gestartet.
Systemname	Preprocessing
Systemverhalten	Tupel aus Annotations- und Quantifizierungs-Objekten werden erzeugt.

2. Persistieren eines Lerndatensatzes

Vorbedingungen	Es ist ein Lerndatensatz aus Tupeln (Quantifizierungen, Annotation) gegeben.
Trigger	Das Programm zum Erzeugen eines Lerndatensatzes wird gestartet.
Systemname	Preprocessing
Systemverhalten	Ein Lerndatensatz wird in einer Datenbank persistiert.

3. Training eines Modells

Vorbedingungen	Ein Lerndatensatz aus Tupeln von (Quantifizierungen, Annotation) ist gegeben. Ein zu trainierendes Modell ist angegeben und konfiguriert.
Trigger	Das Programm zum Trainieren eines Modells wird gestartet.
Systemname	Machine Learning
Systemverhalten	Ein Modell wird mit einem Lerndatensatz antrainiert.

4. Persistieren eines Modells

Vorbedingungen	Ein konfiguriertes und trainiertes Modell ist gegeben.
Trigger	Das Training eines Modells ist abgeschlossen.
Systemname	Machine Learning
Systemverhalten	Das konfigurierte und trainierte Modell wird persistiert.

5. Evaluation eines Modells

Vorbedingungen	Ein dem Trainingsdatensatz x gleichartiger Testdatensatz y ist gegeben. Ein Modell m, trainiert mit x, ist gegeben.
Trigger	Das Programm zur Evaluation des Modell wird gestartet.

Systemname	Machine Learning
Systemverhalten	Die Metriken Accuracy, Precision und Recall werden für jede Klasse aus y des Modells m bestimmt. Es wird die Konfusionsmatrix für m getestet mit y bestimmt.

4.1.2 Anforderungsüberblick: Komponentenrealisierung

In diesem Kapitel werden die Anforderungen an konkrete Implementierungen der Komponenten des Frameworks aufgelistet. Es sollen mithilfe von SonarQube Softwaremetriken von Pfaden von Commits, eines mit Git gemanagten Projektes, gemessen werden. Ein Pfad in einem Commit wird im Folgenden als Commit-Path bezeichnet. Anschließend sollen Dateien anhand der Commit-Message als fehlerbehaftet oder fehlerfrei annotiert werden. Es sollen Metriken über die letzten Änderungen einer Datei mithilfe von SonarQube erzeugt werden. Für jede Lokalität soll die Anzahl der Fehler-Behebungs-indizierenden n zukünftigen Änderungen als Annotation gemessen werden. Die Messungen der nachfolgenden Annotationen und die Messungen der Metriken über die vorangegangenen Änderungen einer Datei sollen zu einem Datensatz mit einem mehrdimensionalen Featurearray, genannt data, sowie einem mehrdimensionalen Annotationsarray, genannt target, zusammengefasst werden. Dieser Datensatz soll sich an den Datensätzen der Softwarebibliothek Scikit-learn orientieren, sodass er für das maschinelle Lernen mittels Scikit-learn einfach verwendet werden kann. Anschließend sollen zwei Machine Learning Modelle gewählt, parametriert, trainiert und evaluiert werden.

1. Erzeugung eines Lerndatensatzes

Vorbedingungen	Eine Datenbank mit den Lokalitäten CommitPaths, Annotationen über die Anzahl der Fehler-Behebungs-indizierenden nächsten fünf Änderungen und Quantifizierungen von Aggregatsfunktionen über Softwaremetriken, gemessen mit SonarQube, der letzten fünf Änderungen einer Datei sind gegeben.
Systemname	commitPath-number-sonarqube-preprocessor-featureSelection

Systemverhalten	Es wird ein Datensatz generiert, welcher die Attribute data und target, konform der Scikit-Learn-Bibliothek, umsetzt. Data und target sind n-dimensionale Arrays aus Zahlen. Jedes Element Data[i] und Target[i] enthält Quantifizierungen und Annotationen zum CommitPath CP(i). Des Weiteren enthält das Dataset das Attribut description, welches eine Beschreibung des Datensatzes enthält, das Attribut featureNames, welches die Bezeichnungen der gemessenen Quantifizierungen enthält, das Attribut targetNames, welches die Bezeichnungen der Annotationen enthält.

1.1 Messen von CommitPaths

Vorbedingungen	Das mit Git gemanagte Open-Source-Projekt TypeScript ist gegeben.
Trigger	Das Programm zum Messen der Lokalitäten wird gestartet.
Systemname	localityRecorder-commitPath
Systemverhalten	Es werden Objekte zu jedem Pfad eines Commits, CommitPaths, erzeugt.

1.2 Persistieren von CommitPaths

Vorbedingungen	CommitPaths sind gegeben.
Trigger	Das Speichern der CommitPaths wird angestoßen.
Systemname	db-mongodb
Systemverhalten	Die CommitPaths-Objekte werden in einer MongoDB persistiert.

1.3 Vorverarbeiten von CommitPaths

Vorbedingungen	CommitPaths-Objekte sind gegeben.
Trigger	Das Programm zum Vorverarbeiten von CommitPaths wird gestartet.

Systemname	localityPreprocessor-commitSubset
Systemverhalten	Eine Teilmenge, CommitPaths aus 10000 Commits, werden gefiltert. Für jeden Commit wird ein CommitPath für das src-Verzeichnis des TypeScript-Projekts erzeugt.

1.4 Speichern der vorverarbeiteten CommitPaths

Vorbedingungen	Vorverarbeitete CommitPath-Objekte sind gegeben.
Trigger	Das Speichern der vorverarbeiteten CommitPaths wird angestoßen.
Systemname	db-mongodb
Systemverhalten	Die vorverarbeiteten CommitPath-Objekte werden in einer MongoDB gespeichert.

1.5a Messen von Softwaremetriken mit SonarQube

Vorbedingungen	Die vorverarbeiteten CommitPaths aus Anforderung 1.3 respektive 1.4 sind gegeben. Das Git-Repository des TypeScript-Projekts ist gegeben.
Trigger	Das Programm zum Messen der Quantifizierungen wird gestartet.
Systemname	quantifier-SonarQube
Systemverhalten	Es werden mithilfe von SonarQube, für jeden gegebenen CommitPath, Softwaremetriken, über die gegebene Datei des Commits, gemessen. Es werden Tupel aus je einem CommitPath und einem SonarQube-Softwaremetrik-Objekt erzeugt.

1.5b Messen von Aggregatsfunktionen über Softwaremetriken vergangener CommitPaths mit SonarQube

Vorbedingungen	Die vorverarbeiteten CommitPaths aus Anforderung 1.3 respektive 1.4 sind gegeben. Das Git-Repository des TypeScript-Projekts ist gegeben.
Trigger	Das Programm zum Messen der Quantifizierungen wird gestartet.
Systemname	quantifier-SonarQubePredecessors
Systemverhalten	Es werden mithilfe von SonarQube, für jeden gegebenen CommitPath, Aggregatsfunktionen über die SonarQube-Softwaremetriken der letzten n Änderungen dieses CommitPaths gemessen. Es werden Tupel aus je einem CommitPath und einem SonarQube-Softwaremetrik-Aggregat-Objekt erzeugt.

1.6a Persistieren der Ergebnisse aus 1.5a

Vorbedingungen	Es sind die Ergebnisse aus 1.5a gegeben.
Trigger	Das Speichern der Ergebnisse aus 1.5a wird angestoßen.
Systemname	db-mongodb
Systemverhalten	Die Ergebnisse aus 1.5a werden in einer MongoDB gespeichert.

1.6b Persistieren der Ergebnisse aus 1.5b

Vorbedingungen	Es sind die Ergebnisse aus 1.5b gegeben.
Trigger	Das Speichern der Ergebnisse aus 1.5b wird angestoßen.
Systemname	db-mongodb
Systemverhalten	Die Ergebnisse aus 1.5b werden in einer MongoDB gespeichert.

1.7a Erzeugung binärer Annotationen anhand der Commit-Beschaffenheit

Vorbedingungen	Die vorverarbeiteten CommitPaths aus Anforderung 1.3 respektive 1.4 sind gegeben.
Trigger	Das Programm zum Erzeugen von Annotationen wird gestartet.
Systemname	annotator-commitMsg
Systemverhalten	Es wird ein Annotations-Objekt zu jedem CommitPath-Objekt erzeugt. Das Annotations-Objekt enthält entweder den Zahlenwert null oder eins. Null steht für: Es wurde keine Fehlerbehebung an der Datei in diesem Commit vorgenommen, eins dafür, dass eine Fehlerbehebung vorgenommen wurde. Zur Berechnung der Annotation soll die Commit-Message nach Fehler indizierenden Schlagwörtern geprüft werden. Es werden Tupel aus je einem CommitPath und einer Annotation erzeugt.

1.7b Erzeugen von Aggregatfunktionen über Annotationen vergangener CommitPaths anhand der Commit-Beschaffenheit

Vorbedingungen	Die vorverarbeiteten CommitPaths aus Anforderung 1.3 respektive 1.4 sind gegeben. Das Git-Repository des TypeScript-Projekts ist gegeben.
Trigger	Das Programm zum Erzeugen von Annotationen wird gestartet.
Systemname	annotator-commitMsgWindow
Systemverhalten	Es werden mithilfe der Komponente zur Realisierung von 1.7a die Summe der n nachfolgenden Fehler-Behebungs-indizierenden Änderungen als Annotation gemessen. Es werden Tupel aus je einem CommitPath und einer Annotation erzeugt.

1.8a Persistieren der Ergebnisse aus 1.7a

Vorbedingungen	Es sind die Ergebnisse aus 1.7a gegeben.
Trigger	Das Speichern der Ergebnisse aus 1.7a wird angestoßen.

Systemname	db-mongodb
Systemverhalten	Die Ergebnisse aus 1.7a werden in einer MongoDB gespeichert.

1.8b Persistieren der Ergebnisse aus 1.7b

Vorbedingungen	Es sind die Ergebnisse aus 1.7b gegeben.
Trigger	Das Speichern der Ergebnisse aus 1.7b wird angestoßen.
Systemname	db-mongodb
Systemverhalten	Die Ergebnisse aus 1.7b werden in einer MongoDB gespeichert.

1.9 Erzeugen eines Lerndatensatzes

Vorbedingungen	Es sind die Ergebnisse aus 1.5b respektive 1.6b gegeben. Es sind die Ergebnisse aus 1.7b respektive 1.8b gegeben. Es ist eine Beschreibung des Datensatzes gegeben. Es sind zu entfernende Metriken (Quantifizierungs-Attribute/Features) gegeben.
Trigger	Das Programm zum Erzeugen eines Lerndatensatzes wird gestartet.
Systemname	preprocessor:featureSelection
Systemverhalten	Es wird der, in der 1. Anforderung beschriebene, Datensatz erzeugt. Im Datensatz befinden sich nicht die zu entfernenden Metriken.

2. Persistieren eines Lerndatensatzes

Vorbedingungen	Es sind die Ergebnisse aus 1.9 gegeben.
Trigger	Das Speichern der Ergebnisse aus 1.9 wird angestoßen.
Systemname	db-mongodb
Systemverhalten	Die Ergebnisse aus 1.9 werden in einer MongoDB gespeichert.

3. Aufteilung in Test- und Trainingsdaten

Vorbedingungen	Der Datensatz aus 1.9 ist gegeben.
Trigger	Das Aufteilen in Test- und Trainingsdaten wird angestoßen.
Systemname	machine-learning
Systemverhalten	70% der Samples des Datensatzes aus 1.9 werden dem Trainingsdatensatz zugeordnet, 30% dem Testdatensatz.

4. Modell-Training

Vorbedingungen	Der Trainings- und Testdatensatz aus 3. ist gegeben.
Trigger	Das Trainieren der Modelle wird angestoßen.
Systemname	machine-learning
Systemverhalten	Es werden zwei Modelle, ein Decision-Tree-Classifier- und ein Random-Forest-Classifier-Modell, mit den Trainingsdaten trainiert.

5. Persistieren der Modell

Vorbedingungen	Ein trainiertes Modell, aus Anforderung 4, ist gegeben.
Trigger	Das Training eines Modells ist abgeschlossen.
Systemname	machine-learning
Systemverhalten	Das trainierte Modell wird als Datei persistiert.

6. Evaluation eines Modells

Vorbedingungen	Der Testdatensatz aus Anforderung 3 ist gegeben. Ein trainiertes Modell aus Anforderung 4 ist gegeben.
Trigger	Die Evaluation des Modells wird angestoßen.
Systemname	machine-learning
Systemverhalten	Die Metriken Accuracy, Precision und Recall werden für jede Klasse des Testdatensatzes bestimmt. Es wird die Konfusionsmatrix für das Modell und den Testdatensatz.

4.1.3 Qualitative Ziele

Die folgende Tabelle beschreibt die zentralen Qualitätsziele des zu entwickelnden Frameworks.

Portabilität: Austauschbarkeit der Komponenten

Ursprung	Forscher
Stimulus	Möchte andere Komponenten verwenden
Umgebung	Design-Zeit
Artefakt	Komponenten des Frameworks: LocalityRecorder, LocalityPreprocessor, Quantifier, Annotator und Preprocessor. Einzutauschende Komponente.
Antwort	Einzutauschende Komponente wird als zu verwendende Komponente in der Konfiguration der Dependency Injection gesetzt.
Antwort-Messung	Neu angegebene Komponente wird verwendet.

Portabilität: Installierbarkeit der Komponenten

Ursprung	Forscher

Stimulus	Möchte eigene Komponenten mit anderen teilen und Komponenten anderer nutzen.
Umgebung	Design-Zeit
Artefakt	Eine zu installierende Komponente für entweder LocalityRecorder, LocalityPreprocessor, Quantifier, Annotator oder Preprocessor.
Antwort	Komponente wird als Datei(en) weitergegeben. Zum Austausch der Dateien wird Github und npm verwendet.
Antwort-Messung	Download der Datei(en) erfolgt in unter fünf Minuten.

Austauschbarkeit des Lokalitätstypus

Ursprung	Forscher
Stimulus	Möchte neuen Typ von Lokalität einführen oder nutzen
Umgebung	Design-Zeit
Artefakt	Neue Lokalitätsklasse, Lokalitätsschnittstelle
Antwort	Eine, der Lokalitätsschnittstelle konforme, Lokalitätsklasse wird eingeführt.
Antwort-Messung	Es müssen keine Schnittstellen des Frameworks angepasst werden. Damit müssen die Schnittstellen des Frameworks die qualitative Anforderung Kompatibilität mit verschiedenen Lokalitätstypen erfüllen.

4.1.4 Stakeholder

Die folgende Tabelle stellt die Stakeholder des Frameworks und ihre jeweilige Intention dar.

Rolle	**Nutzung, Erwartung**
Forscher	Entwickelt bestehende Software weiter, misst neue Trainingsdaten; wählt, trainiert und evaluiert neue und bestehende Modelle; tauscht Entwicklungsartefakte mit anderen Forschern aus; und studiert die Dokumentation des Frameworks und bestehender Komponenten anderer.
Softwareentwickler: Anwender	Verwendet bereits entwickelte Fehlervorhersage-Komponente zur Stellenidentifikation mit erhöhter Fehlerwahrscheinlichkeit. Erwartet einfache Nutzung mit angemessener Performanz für das eigene Softwareprojekt. Erwartet eine Anleitung mit konkretem Beispiel zur Nutzung des Produkts.
Qualitätssicherungs-Mitarbeiter: Anwender	Konfiguriert und integriert eine Fehlervorhersage-Komponente in den Entwicklungsprozess und reagiert individuell auf die Vorhersage eines wahrscheinlichen Fehlerauftretens in neu hinzugefügte Lokalitäten. Nutzt bei Bedarf die Fehlervorhersage-Komponente zur Prüfung einzelner Stellen. Erwartet eine einfache Installation, Integration und Nutzung der Fehlervorhersage-Komponente für die zu untersuchenden Softwareprojekte. Erwartet eine Anleitung zur Installation und Integration mit Beispielen zur Nutzung konkreter Implementierungen einzelner Komponenten und einer geeigneten Verkettung dieser zum Zwecke der Fehlerprädiktion.

4.2 Randbedingungen

Beim Lösungsentwurf waren nur begrenzt Randbedingungen zu beachten. Die folgenden Randbedingungen wurden teilweise aus Rücksicht auf potenzielle Stakeholder gesetzt, sind allerdings durch keine Verbindlichkeit erzwungen.

4.2.1 Technische Randbedingungen

Randbedingung	Erläuterung und Hintergrund
Fremdsoftware frei verfügbar	Verwendete Fremdsoftware sollte frei verfügbar sein, damit ein öffentlicher Austausch dieser Arbeit mit möglichst wenigen Hindernissen ermöglicht werden kann.
Betrieb auf verschiedenen Betriebssystemen (Linux, Windows, MacOS)	Der Betrieb auf verschiedenen Betriebssystemen sollte möglich sein, da auf diesen Software entwickelt und getestet wird, aber auch da Forscher unterschiedliche Präferenzen hinsichtlich der Nutzung bestimmter Betriebssysteme besitzen und deren Interessen berücksichtigt werden sollten.

4.2.2 Organisatorische Randbedingungen

Randbedingung	Erläuterung und Hintergrund
Veröffentlichung als Open Source	Alle Artefakte dieser Arbeit, also die textuelle Ausarbeitung und Dokumentation, Aufgabenstellung und jeglicher Quellcode sollen Open Source zur Verfügung gestellt werden. Dies soll einen einfachen Austausch und die Nutzung dieser Arbeit unter Forschern anregen.
Versionsverwaltung	Es soll Git mit Github verwendet werden, da dieses Programm und diese Plattform eine hohe Reichweite und Verbreitung unter den Stakeholdern besitzt.

4.3 Kontextabgrenzung

Die wichtigsten, äußeren Interaktionen mit dem System, sowie deren Schnittstellen verdeutlichen den Kontext, in dem das System eingesetzt wird. Dieser Rahmen zeigt, welche Ziele das System umsetzt, aber auch wie Akteure das System nutzen können.

4.3.1 Fachlicher Kontext

Forscher nutzen das System, um Softwareprojekte zu quantifizieren, zu annotieren und um Datensätze zu generieren, die geeignet für das Machine Learning sind. Anwender nutzen das System, um Softwareprojekte auf Fehler zu prüfen.

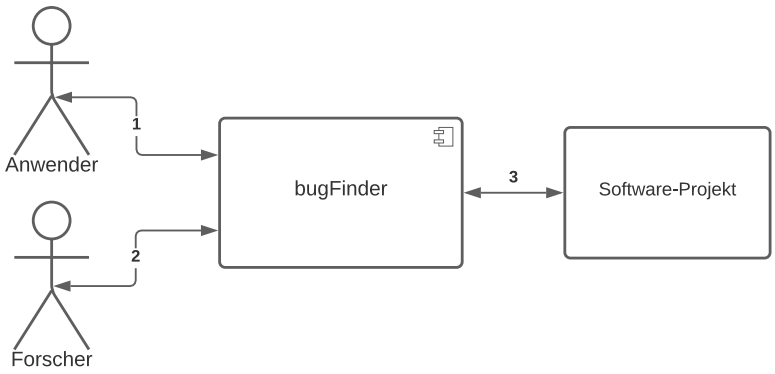

Abbildung 4.1: Fachlicher Kontext

4.3.2 Technischer Kontext

Das bugFinder-System misst Lokalitäten aus einem Git-Repository und stößt SonarQube dazu an, Softwaremetriken aus diesem zu extrahieren. SonarQube bietet, nach erfolgreicher Analyse, die gemessenen Softwaremetriken über ein Webinterface an, welche durch das bugFinder-System abgerufen werden.

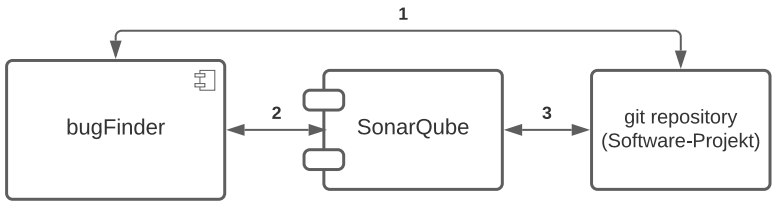

Abbildung 4.2: Technischer Kontext

4.4 Lösungsstrategie

Zur Realisierung des BugFinder-Programms wurde die aus den Grundlagen erklärte Mustererkennungs-Pipeline (siehe Kapitel 3.1.14, S. 20) in Kombination mit der Blackboard-Architektur (siehe Kapitel 3.2, S. 22) als Basis verwendet.

Auf der Abbildung 4.6 auf Seite 60) sind links die Control-Komponenten abgebildet: Recording und Preprocessing, rechts befinden sich die Knowledge-Source-Komponenten: LocalityRecorder, LocalityPreprocessor, Annotator, Quantifier und Preprocessor. Die Schritte der Pipeline werden durch die Recording-, Preprocessing und Machine-Learning-Komponente realisiert. Die Kommunikation der Pipelineschritte erfolgt über das Blackboard. Die Recording-Komponente sammelt Daten, welche durch die Preprocessing-Komponente verarbeitet werden. Durch

die Vorverarbeitung werden die Daten in ein passendes Format für Lerndaten für die Machine-Learning-Komponente gebracht. Die Machine-Learning-Komponente dient dem Extrahieren von Wissen aus den Daten, sodass auf Basis gleichartiger Quantifizierungs-Daten Vorhersagen über die Annotationen getroffen werden können.

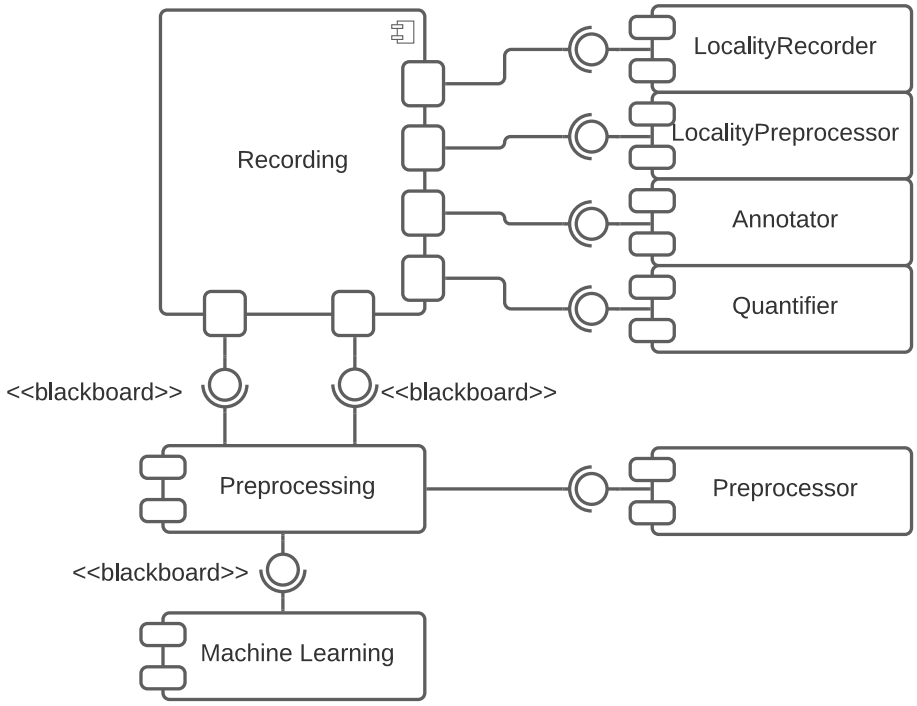

Abbildung 4.3: Lösungsstrategie

Verwendete Technologien
Verschiedene Technologien werden verwendet, um einzelne Komponenten zu realisieren und um bestimmten Softwarequalitätsanforderungen zu adressieren.

Alle Komponenten, mit Ausnahme der Machine-Learning-Komponente, werden in TypeScript realisiert. Es wird Dependency Injection genutzt, um Austauschbarkeit der Knowledge-Source-Komponenten sicherzustellen (Ref. Qualitative Ziele: Austauschbarkeit der Komponenten). Realisierungen der Knowledge-Source-Komponenten werden als npm-Pakete in der npm-Registry und auf Github hochgeladen, sodass sie mittels npm install zügig installiert werden können (Ref. Qualitative Ziele: Installierbarkeit der Komponenten). Die Machine-Learning-Komponente wird in Python, unter Verwendung typischer Machine-Learning-Bibliotheken wie Scikit-Learn, als Template, primär in einer Jupyter-Notebook-Datei veröffentlicht, entwickelt. Es wird Anaconda zur Erzeugung einer virtuellen Umgebung für das Machine-Learning-Projekt verwendet.

4.5 Bausteinsicht

Im Folgenden wird die statische Dekomposition der einzelnen Komponenten aufgeführt und erläutert. Die Komponente Recording bildet das Sammeln von Daten ab. Die Komponente Preprocessing verwendet Daten der Recording-Komponente, konkret die quantifizierten und die annotierten Lokalitäten, um sie in ein standardisiertes Format zu bringen, welches anschließend von der Machine-Learning-Komponente verwendet wird, um maschinelle Lernverfahren anzuwenden.

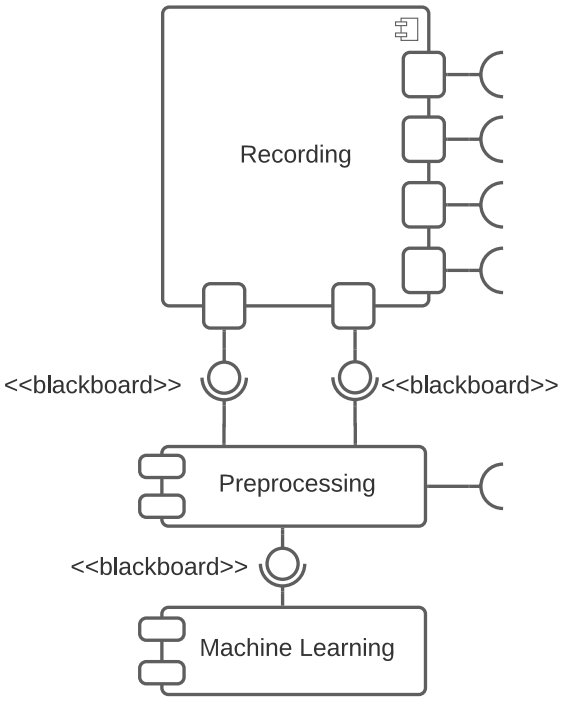

Abbildung 4.4: Top-Level-Dekomposition

4.5.1 Recording

Die Recording-Komponente ist aufgeteilt in vier weitere Komponenten: Die LocalityRecording-Komponente, welche für das Aufzeichnen von Lokalitäten zuständig ist, die LocalityPreprocessing-Komponente, welche für die Vorverarbeitung der Lokalitäten zuständig ist, die Annotating- und Quantifying-Komponenten, welche die vorverarbeiteten Lokalitäten nutzen, um eine Annotation und eine Quantifizierung für jede Lokalität zu erzeugen. Die LocalityRecording-Komponente benötigt eine Komponente, welche Lokalitäten aufzeichnet, die LocalityPreprocessing-Komponente benötigt eine Komponente, welche Lokalitäten vorverarbeitet, die

Annotating-Komponente benötigt eine Komponente, welche Lokalitäten annotiert, die Quantifying-Komponente, benötigt eine Komponente, welche Lokalitäten quantifiziert, die Preprocessing-Komponente benötigt eine Komponente welche quantifizierte und annotierte Lokalitäten in einen Lerndatensatz transformiert.

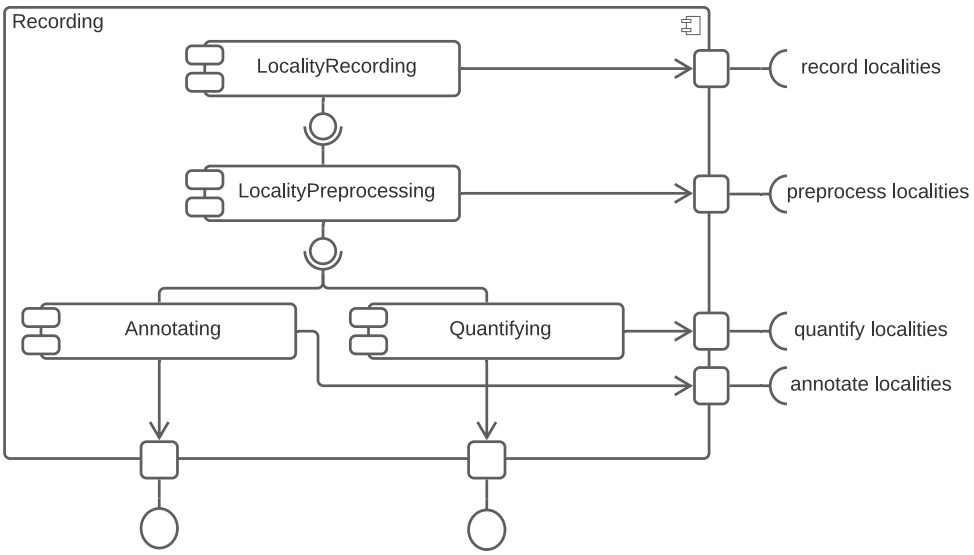

Abbildung 4.5: Komponente: Recording

4.5.2 Machine-Learning

Die Machine-Learning-Komponente enthält eine Manager-Komponente, welche alle anderen Komponenten des Machine-Learnings verwaltet und verwendet. Die Standardisierungs/Normalisierungs-Komponente übt Standardisierung und Normalisierung auf Daten aus. Die Feature-Extraction Komponente übt Feature-Extraction auf Daten aus. Die Model-Generator-Komponente erzeugt Machine-Learning-Modelle. Die Evaluator-Komponente evaluiert Machine-Learning-Modelle.

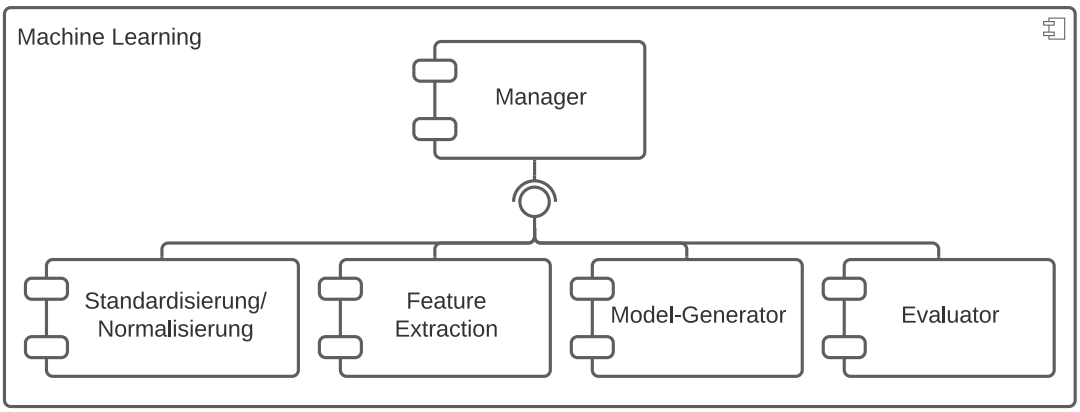

Abbildung 4.6: Komponente: Machine Learning

4.6 Laufzeitsicht

Die Ziele dieser Arbeit sind in zwei Teile zu differenzieren: Der Lern-Phase und die Prädiktions-Phase. Der Fokus dieser Arbeit liegt auf der Lernphase, wenngleich unter Verwendung der Schritte der Lernphase und der Anwendung der Ergebnisse des `learn`-Schrittes damit praktisch auch die (unoptimierte) Prädiktionsphase gegeben ist. Weiterentwicklungen respektive Verwendungen dieser Arbeit ermöglicht die Realisierung der Prädiktionsphase. Das Programm, das die Prädiktionsphase realisiert, ist ein automatisierter Bugprädizierer.

Die hier dargestellte Architektur ist eine Pipeline. Eingaben der vorangehenden Aktivitäten werden verarbeitet, um Ausgaben für die nachfolgenden Aktivitäten zu generieren. Lediglich die Schritte Quantifizieren und Annotieren verlaufen parallel. Jede dargestellte Aktivität der Lernphase ist ein Prozess, bei welchem die Ergebnisse in einer Datenbank gespeichert und in der anschließenden Aktivität wieder gelesen werden.
In den Schritten *record localities*, *preprocess localities*, *quantify* und *annotate* werden Daten gesammelt und erzeugt.

Record localities: Konkret werden Lokalitäten, Stellen, die quantifizierbar und annotierbar sind, erzeugt. Beispiele für Lokalitäten sind Commits und Commit-Paths, Pfade in einem Commit.

Preprocess localities: Die gemessenen Lokalitäten werden anschließend vorverarbeitet. Mögliche Vorverarbeitungprozesse sind das Filtern von Lokalitäten, das Anreichern der Lokalitäten mit Daten oder das Injizieren weiterer Lokalitäten. Auch keine Vorverarbeitung ist eine mögliche Vorverarbeitung. Die Ergebnisse der Lokalitäten-Vorverarbeitungs-Aktivität sind Lokalitäten.

Quantify: Diese werden für das Quantifizieren und Annotieren verwendet. Beim Quantifizieren werden einer Lokalität eins bis n Quantitäten zugeordnet. Eine

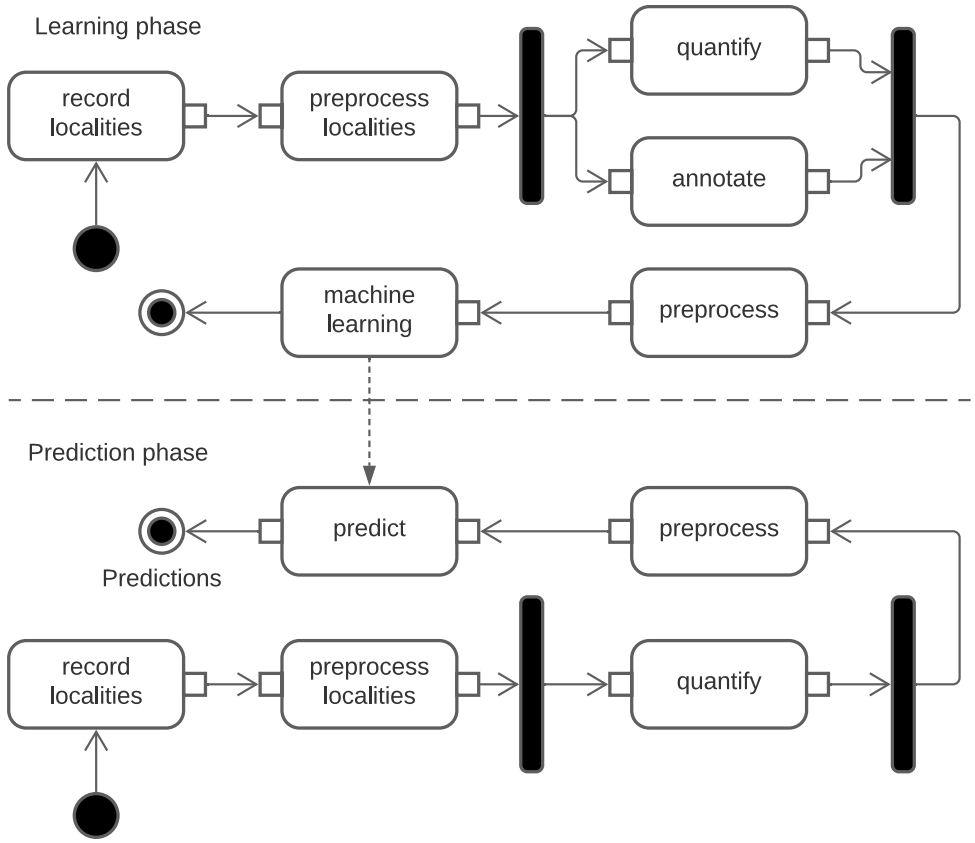

Abbildung 4.7: Pipeline

Quantität ist ein numerischer Wert. Ein Beispiel für das Quantifizieren eines CommitPaths ist das Messen von Softwaremetriken.

Annotate: Beim Annotieren werden einer Lokalität ebenfalls eins bis n Quantitäten zugeordnet, den Annotationen. Eine Annotation ist die Zielgröße, die vorhergesagt werden soll. Für das Überwachte Lernen (Supervised Learning) müssen Samples (Quantitäten je Lokalität) mit den bereits bekannten Zielgrößen gegeben werden. In der Prädiktionsphase ist diese Zielgröße nicht mehr vorhanden, sie wird durch Ergebnis des Machine-Learnings, einem trainierten Modell, prädiziert (vorhergesagt).

Preprocess: Anschließend werden sie in der Aktivität preprocess in ein Datenformat gebracht, das für Machine-Learning-Algorithmen geeignet ist.

Learn: Die learn-Aktivität bezeichnet das maschinelle Lernen ohne Datenerfassung. Darunter fallen: Daten-Normalisierung und Skalierung, Feature-Selection und -Extraction, Aufteilung in Trainings- und Testdatensatz, Modell- und Hyperparamterwahl und Training. Die *Learn*-aktivität ist ein manueller Schritt der Pipeline, welcher durch einen Domänenexperten bearbeitet werden sollte. Sie wird

in dieser Arbeit als Template gegeben. Die Ergebnisse der *Learn*-Aktivität dienen der *Predict*-Aktivität der Prädiktionsphase.

4.7 Verteilungssicht

Alle Komponenten, zur Realisierung der Schritte der Lernphase aus 4.7 auf Seite 61, werden in einzelne JavaScript-Dateien transpiliert und manifestiert. Sie alle können in einer NodeJS-Ausführungsumgebung ausgeführt werden.

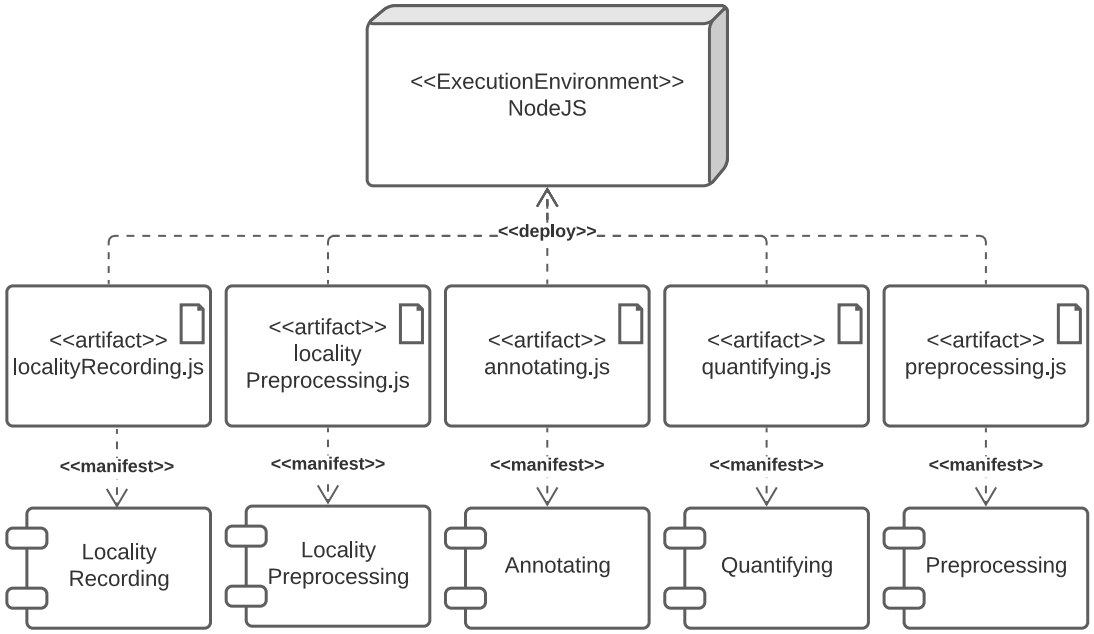

Abbildung 4.8: Verteilungssicht

Die Implementierung des Quantifizierers mittels SonarQube und der Datenbank mittels MongoDB ermöglicht die Verteilung aus Abbildung 4.9: Der Computer kommuniziert mit einem SonarQube-Server, um Softwaremetriken abzurufen, sowie mit einer MongoDB, welche als Blackboard fungiert. Der SonarQube-Server und die MongoDB können sich auf einem beliebigen erreichbaren Computer befinden. Im Rahmen dieser Arbeit befand sich der SonarQube-Server, die MongoDB und die NodeJS-Umgebung auf einem PC. Das Quantifizieren wurde horizontal skaliert, indem alle diese Knoten vervielfältigt wurden, also mehrere Computer verwendet wurden, auf welchen sich jeweils eine MongoDB, ein SonarQube-Server und eine NodeJS-Ausführungsumgebung befanden. Alle Computer führten dann auf einen eigenen Teil an Lokalitäten die Quantifizierung aus und die Daten wurden anschließend zusammengetragen. MongoDB soll es zukünftigen Arbeiten ermöglichen, verteilt auf mehreren Computern zu rechnen und direkt alle

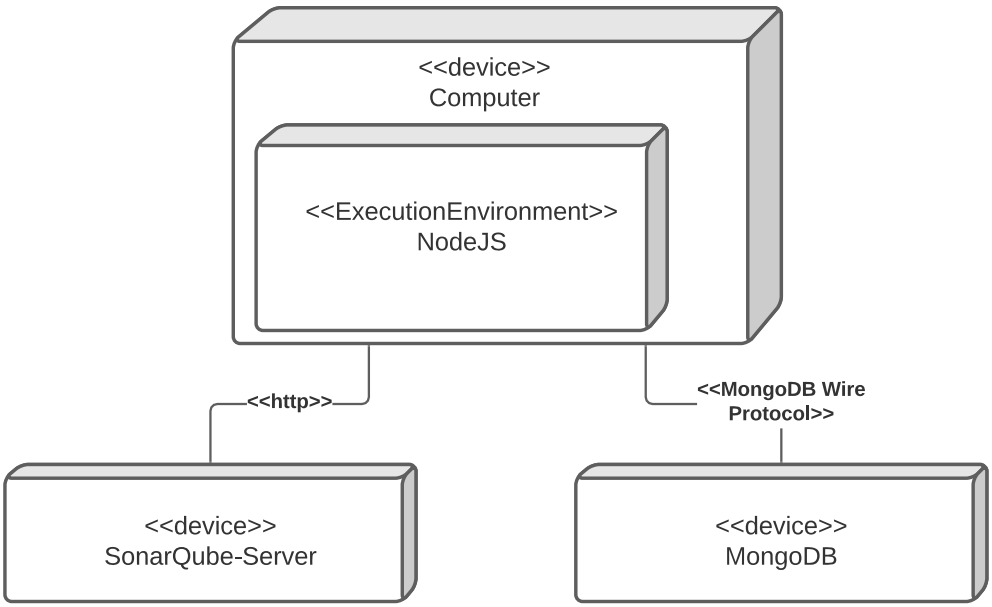

Abbildung 4.9: Horizontale Skalierung: Quantifizierer

Ergebnisse in einer Datenbank zusammenzutragen.

4.8 Konzepte

Mit dem Ziel bestimmte qualitative Ziele zu adressieren werden bestimmte Konzepte verwendet.

4.8.1 Dependency Injection

Es wird Dependency Injection verwendet, um wichtige Komponenten austauschbar zu gestalten und um Plugins konfigurierbar zu gestalten.

4.8.2 Npm-Pakete und Github

Alle Komponenten werden als öffentlich zugängliche git-Repositories angelegt. Alle Plugins werden als npm-Pakete realisiert, welche in der npm-Registry registriert sind. Dies ermöglicht eine standardkonforme, zügige Installation, sowie eine einfach Auffindbarkeit dieser Pakete.

4.8.3 Paket-Namen

Die Plugin-Pakete-Namen erfüllen eine strikte Reihenfolge, in denen gegebene Typ-Abhängigkeiten abgebildet werden. Alle Plugins besitzen, zusätzlich zur Ver-

wendung des Präfixes bugfinder, folgende Namenskonventionen für die Realisierung der entsprechenden Komponente:

Plugin-Komponente	Name	Beispiel
LocalityRecorder	localityrecorder-TYP	localityrecorder-commitpath
LocalityPreprocessor	localitypreprocessor-$LOKALITÄT$-$name$	localitypreprocessor-commitpath-commitsubset
Quantifizierer	$LOKALITÄT$-quantifier-$name$	commitpath-quantifier-sonarqube
Annotierer	$LOKALITÄT$-annotator-$name$	commitpath-annotator-commitmsg
Preprocessor	$LOKALITÄT$-$ANNOTATIONS-TYP$-$QUANTIFIZIERUNGS-TYP$-$name$	commitpath-number-sonarqube-preprocessor-featureSelection

Diese Namenskonventionen sollen helfen alle passenden Plugins in der npm-Registry aufzufinden, die für die gewünschte Zwecke genutzt werden können.

4.8.4 Blackboard

Alle Pipeline-Schritte arbeiten auf ein Blackboard, einer gemeinsam geteilten Datenbank. Das Blackboard dient der Memoization, also der Speicherung von Ergebnissen zur Vermeidung derer erneuter Berechnung.

4.9 Plugin Schnittstellen Dokumentation

Für die Komponenten LocalityRecording, LocalityPreprocessing, Annotating, Quantifying und Preprocessing wurden Schnittstellen-Definitionen im Paket *bugFinder-Framework* zur Verfügung gestellt. Diverse Pakete implementieren einzelne Komponenten der im bugFinder-Framework-Paket definierten Schnittstellen. Im Folgenden ist unter Paket, auch in den Abbildungen, ein npm-Paket gemeint. Die dargestellten Komponenten beziehen sich in diesem Kapitel auf die Unterteilung des bugFinder-Framework-Pakets und nicht auf die Komponenten der Lösungsstrategie. Jedes implementiernde Paket kann als Plugin in den Kontroll-Komponenten verwendet werden.

4.9.1 Paket: Framework

Das Framework-Paket enthält alle Schnittstellen, die für die Realisierung der einzelnen Schritte des Recording und Preprocessing benötigt werden (siehe Abb. 4.10).

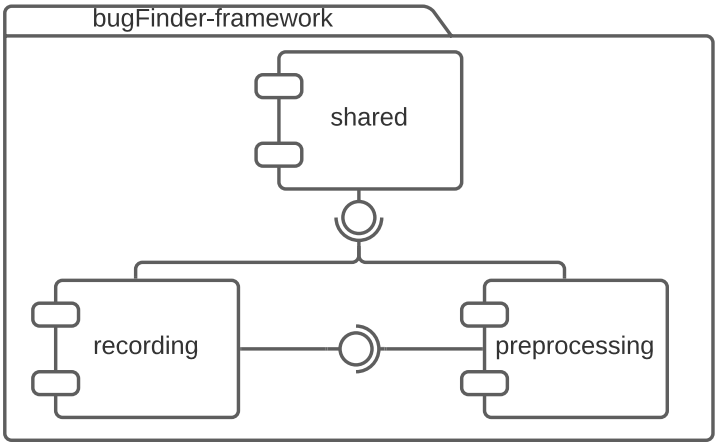

Abbildung 4.10: Framework-Paket

Darüber hinaus enthält es eine geteilte Komponente, die shared-Komponente. Mittels Dependency Injection können Typen der einzelnen Pakete injiziert werden.

4.9.2 Dependency Injection mit InversifyJS

Für die Dependency Injection mit InversifyJS werden sogenannte TYPES verwendet. Types sind keys, die zum Auflösen des zu injizierenden Wertes verwendet werden. Das bugFinder-Framework-Paket bietet für jede Komponente und damit für jeden Prozess aus bugFinder eigene TYPES, welche die benötigten Typen für den Prozess angeben, an.

Beispiel
Typen für das Sammeln von Lokalitäten, entnommen aus bugFinder-Framework:

```
1 export const LOCALITY_RECORDING_TYPES = {
2     localityRecorder:          Symbol("LocalityRecorder"),
3     db:                        Symbol("DB"),
4     localityRecordingFactory:  Symbol("LocalityRecordingFactory")
5 }
```

Analog gibt es die Types `LOCALITY_PREPROCESSING_TYPES`, `QUANTIFIER_TYPES`, `ANNOTATOR_TYPES`, `PREPROCESSING_TYPES`, `TRAINING_TYPES`, `PREDICTION_TYPES`, sowie `SHARED_TYPES` für unter den realisierenden Paketen geteilte injizierbare Typen.

Alle anderen Pakete können zur Konfiguration des eigenen Pakets und im Sinne der Austauschbarkeit interner Komponenten ebenfalls InversifyJS benutzt. Als Konvention heißen deren Types: `$Paket-Name$_TYPES`. Jedes bereits realisierte und veröffentlichte Paket enthält eine Dokumentation der zu bindenen Typen, sowie bindbare Beispielwerte. Zur Nutzung eines Pakets kann die Angabe einer Konfiguration notwendig sein. Sie finden eine Anleitung zu Dependency Injection mit InversifyJS in Anhang A auf Seite 124.

4.9.3 Komponente: Shared

Die shared-Komponente stellt einen Logger und eine Datenbankschnittstelle bereit (siehe S. 67 Abb. 4.11). Der FileAndConsoleLogger ist eine Implementierung der mittels npm öffentlich zugänglichen ts-log-Paket-Logger-Schnittstelle.

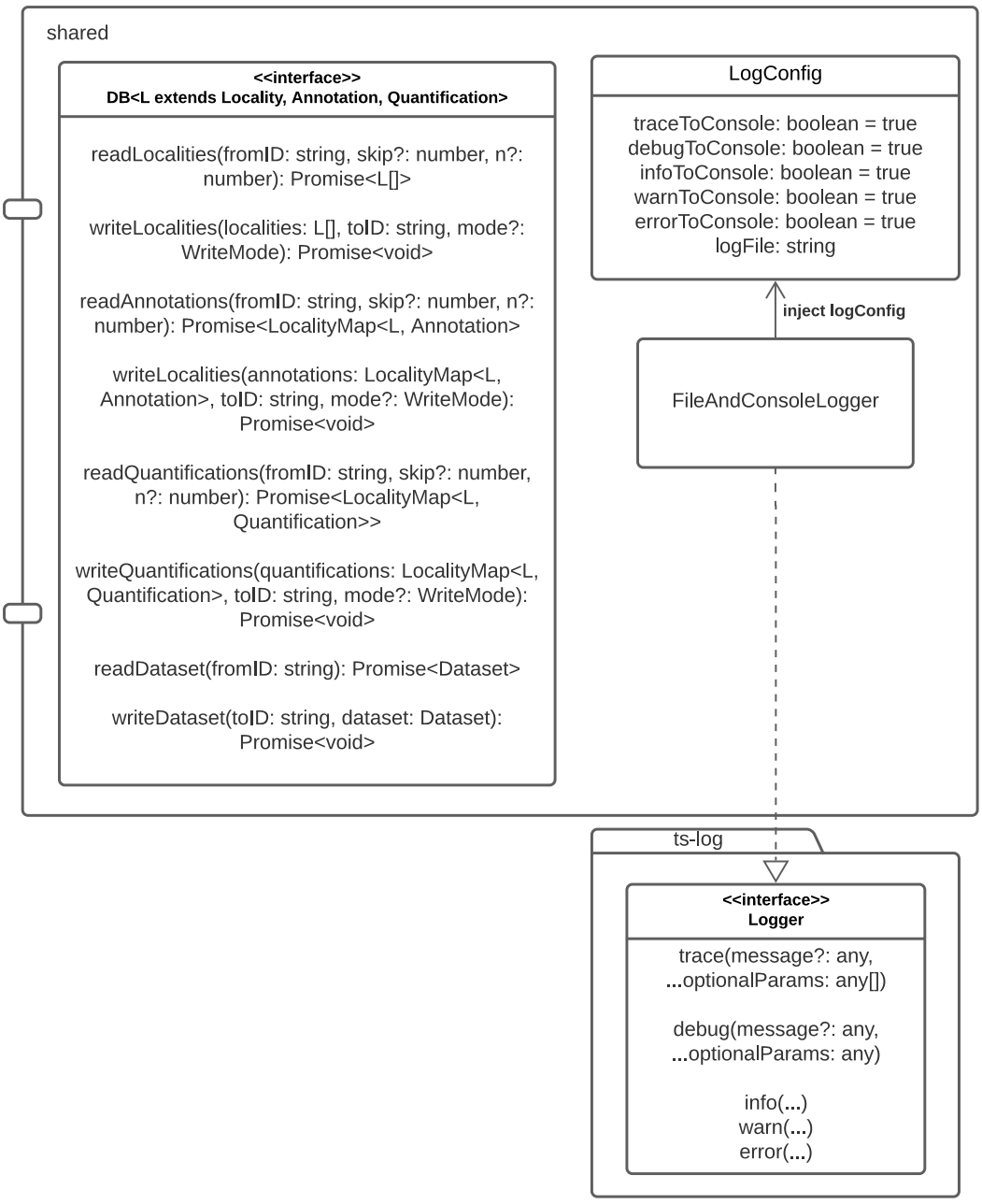

Abbildung 4.11: shared-Komponente

Die Types der shared-Komponente heißen: `SHARED_TYPES`. Sie ermöglichen das Injizieren eines Loggers und der LogConfig. Viele Pakete nutzen diesen injizierten Logger, um Warnungen, mögliche Fehler oder Informationen zu loggen. Da einige Prozesse der Datenerhebung viel Zeit in Anspruch nehmen und es wahrscheinlich ist, dass im Verlauf der Messung viele Sonderfälle auftreten, hat sich der Logger als bewährte Maßnahme etabliert, die Fehlertoleranz zu erhöhen.

Name	SHARED_TYPES	
Attribut	**Typ**	**Beispiel**
logger	ts-log-logger	FileAndConsoleLogger
logConfig	LogConfig	`{ traceToConsole: true, debugToConsole: false, infoToConsole: false, warnToConsole: true, errorToConsole: true, logFile: "./log.txt" }`

4.9.4 Komponente: Recording

Die Recording-Komponente besteht aus mehreren Subkomponenten: localityRecording, localityPreprocessing, quantifications und annotations, welche die einzelnen Schritte der Datenerhebung abbilden.

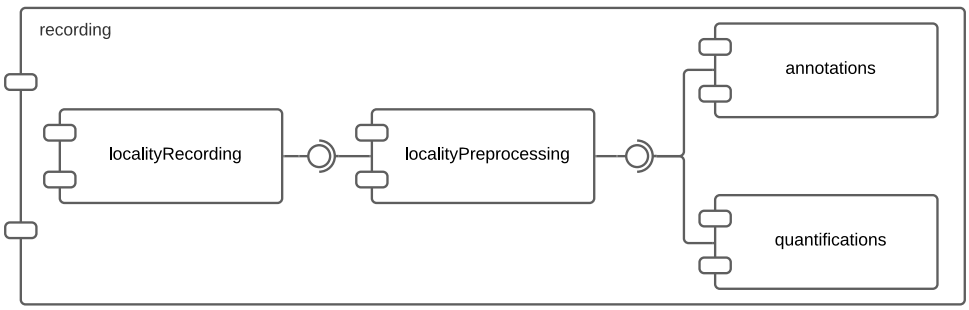

Abbildung 4.12: Framework: Recording

Komponente: LocalityRecording

Die LocalityRecording-Komponente bildet das Sammeln/Erheben der Lokalitäten ab, die anschließend quantifiziert und annotiert werden sollen. Die Implementierung der Schnittstelle `LocalityRecorder` realisiert das Sammeln der Lokalitäten. Eine Lokalität implementiert die Schnittstelle `Locality`, sie realisiert eine Funktion `key`, welche einen einzigartigen Schlüssel für diese Lokalität berechnet. Dieser Schlüssel wird in der Wrapper-Klasse `LocalityMap` genutzt.

Die `LocalityMap` ist eine Map, welche von Lokalitäten auf Werte abbildet. Sie wird z.B. genutzt, um von Lokalitäten auf Annotationen oder von Lokalitäten auf Quantifizierungen abzubilden. Sie dient der Performanz und der Möglichkeit der Persistierung dieser Map. Eine Map mit keys aus komplexen Datentypen ist nur zur Laufzeit des Programms gültig, da die Speicheradressen der Lokalitäten nach jedem Start des Programms in der Regel anders ausfallen.
Eine `Locality` besitzt einen optionalen `parentKey`, welcher der Normalisierung hierarchischer Lokalitäten dienen soll.
Zur Erzeugung eines mit einer `Datenbank` zusammenpassenden `LocalityRecorder` dient die `LocalityRecordingFactory`. Mittels der durch die `LocalityRecordingFactory` gegebenen Instanzen können zunächst Lokalitäten erzeugt und anschließend gespeichert werden.

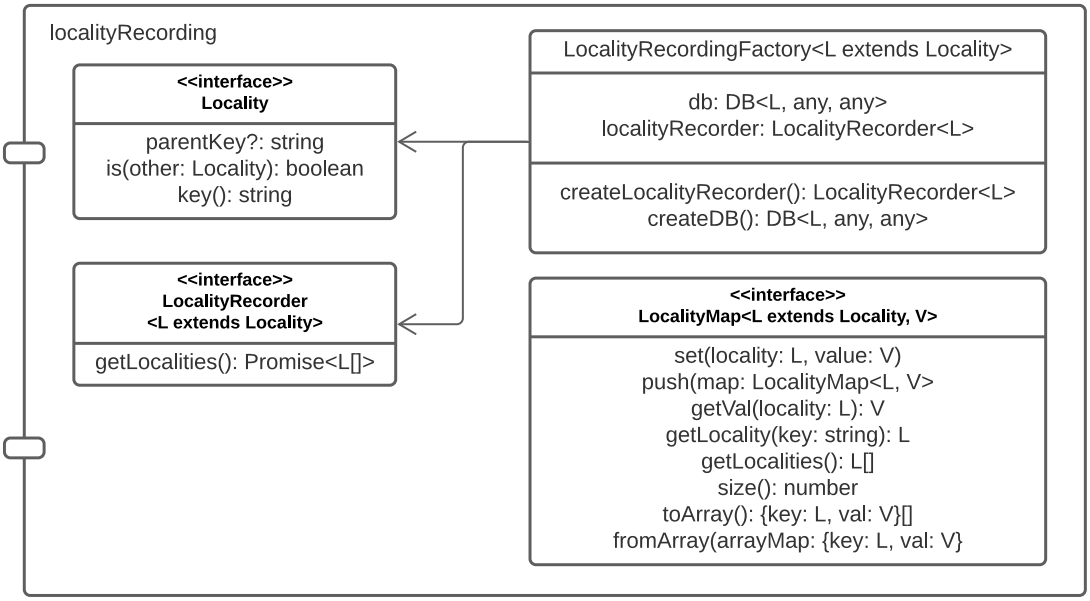

Abbildung 4.13: Framework: Recording: LocalityRecording

Die Types der localityRecording-Komponente heißen: `LOCALITY_RECORDING_TYPES`. Sie ermöglichen das injizierten eines `LocalityRecorders`, einer Datenbank `DB`, sowie einer `LocalityRecordingFactory`.

Name	LOCALITY_RECORDING_TYPES	
Attribut	**Typ**	**Beispiel**
localityRecorder	LocalityRecorder<CommitPath>	CommitPathRecorder
db	DB<CommitPath, any, any>	CommitPathsMongoDB

localityRecording Factory	LocalityRecordingFactory <CommitPath>	LocalityRecordingFactory

Komponente: LocalityPreprocessing

Die LocalityPreprocessing-Komponente bildet das Vorverarbeiten der Lokalitäten ab. Es können Lokalitäten gefiltert oder bearbeitet werden, falls nötig. Die Implementierung der Schnittstelle `LocalityPreprocessig` realisiert das Vorverarbeiten der Lokalitäten. Zur Erzeugung eines mit einer Datenbank `DB` zusammenpassenden `LocalityPreprocessors`, dient die `LocalityPreprocessingFactory`.

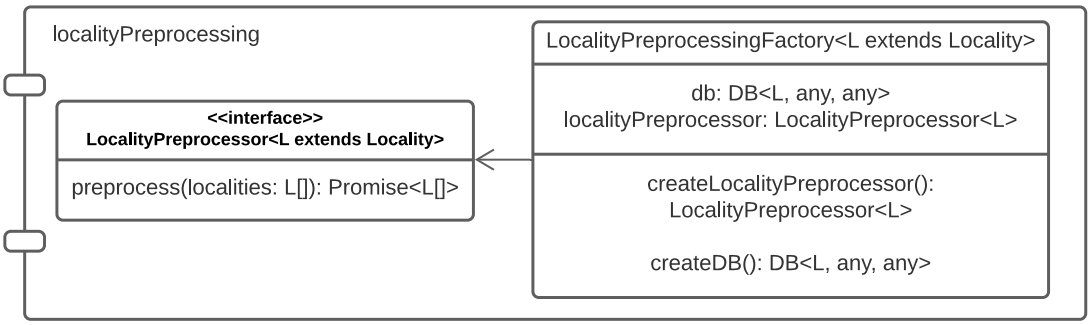

Abbildung 4.14: Framework: Recording: LocalityPreprocessing

Die Types der localityPreprocessing-Komponente heißen:
`LOCALITY_PREPROCESSING_TYPES`. Sie ermöglichen das injizierten eines `LocalityPreprocessors`, einer Datenbank `DB`, sowie einer `LocalityRecordingFactory`.

Name	LOCALITY_PREPROCESSING_TYPES	
Attribut	**Typ**	**Beispiel**
localityPreprocessor	LocalityPreprocessor<CommitPath>	CommitSubset
db	DB<CommitPath, any, any>	CommitPathsMongoDB
localityPreprocessing Factory	LocalityPreprocessingFactory <CommitPath>	LocalityPreprocessing Factory

Komponente: Quantification

Die Quantification-Komponente bildet das Quantifizieren der Lokalitäten ab. Die Implementierung der Schnittstelle `Quantifier` realisiert das Quantifizieren der Lokalitäten. Es wird eine Map von Lokalitäten auf Quantifizierungen gebildet. Zur Erzeugung eines mit einer Datenbank `DB` zusammenpassenden `Quantifiers`, dient die `QuantificationFactory`.

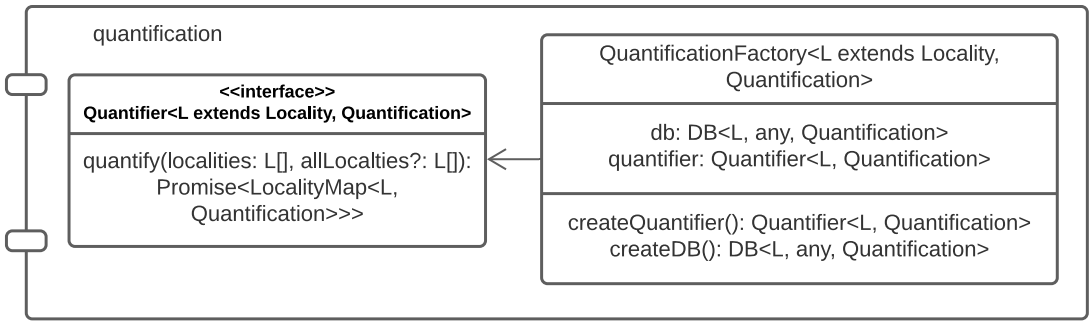

Abbildung 4.15: Framework: Recording: Quantification

Die Types der Quantification-Komponente heißen: `QUANTIFIER_TYPES`. Sie ermöglichen das injizierten eines `Quantifier`, einer Datenbank `DB`, sowie einer `QuantificationFactory`.

Name	QUANTIFIER_TYPES	
Attribut	**Typ**	**Beispiel**
quantifier	Quantifier<CommitPath>	SonarQubeQuantifier
db	DB<CommitPath, any, SonarQubeMeasurement>	CommitPathsMongoDB
QuantificationFactory	QuantificationFactory<CommitPath, SonarQubeMeasurement>	QuantificationFactory

Komponente: Annotation

Die Annotation-Komponente bildet das Annotieren der Lokalitäten ab. Die Implementierung der Schnittstelle `Annotator` realisiert das Annotieren der Lokalitäten. Es wird eine Map von Lokalitäten auf Annotationen gebildet. Zur Erzeugung eines mit einer Datenbank `DB` zusammenpassenden `Annotators`, dient die `AnnotationFactory`.

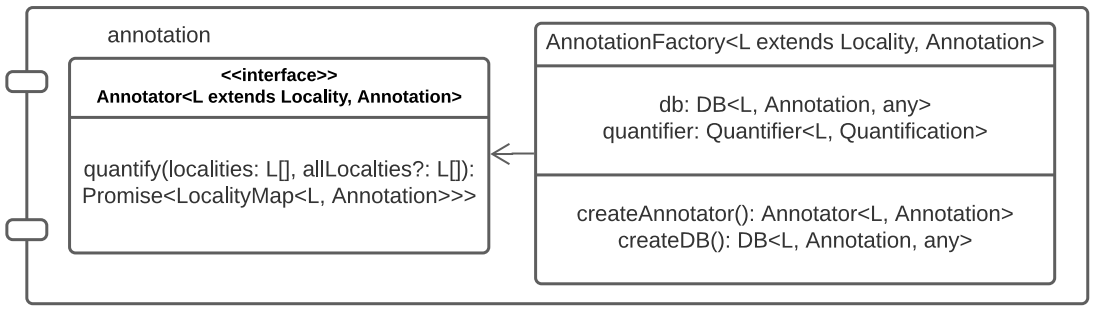

Abbildung 4.16: Framework: Recording: Annotation

Die Types der Annotation-Komponente heißen: `ANNOTATOR_TYPES`. Sie ermöglichen das injizierten eines
`Annotator`s, einer Datenbank `DB`, sowie einer
`AnnotationFactory`.

Name	QUANTIFIER_TYPES	
Attribut	**Typ**	**Beispiel**
annotator	Annotator<CommitPath>	CommitPathsWindow Annotator
db	DB<CommitPath, number, any>	CommitPathsMongoDB
AnnotationFactory	AnnotationFactory<CommitPath, number>	AnnotationFactory

4.9.5 Komponente: Preprocessing

Die Preprocessing-Komponente bildet das Verarbeiten aller Quantifizierungen und Annotationen, sowie das Bilden eines `Datasets` ab. Ein `Dataset` ist ein Format, welches konform der sklearn-Datasets in Python ist. Sie enthalten zusätzlich die Lokality-Keys, sowie Trace-Informationen zur Erzeugung des Datensatzes. Die Implementierung der Schnittstelle `Preprocessor` realisiert das Verarbeiten der Quantifizierungen und Annotationen, sowie das Bilden eines `Dataset`s. Zur Erzeugung eines mit einer Datenbank `DB` zusammenpassenden `Preprocessors`s, dient die `PreprocessingFactory`.

Die Types der Preprocessing-Komponente heißen: `PREPROCESSING_TYPES`. Sie ermöglichen das injizierten eines `Preprocessors`s, einer Datenbank `DB`, sowie einer `PreprocessingFactory`.

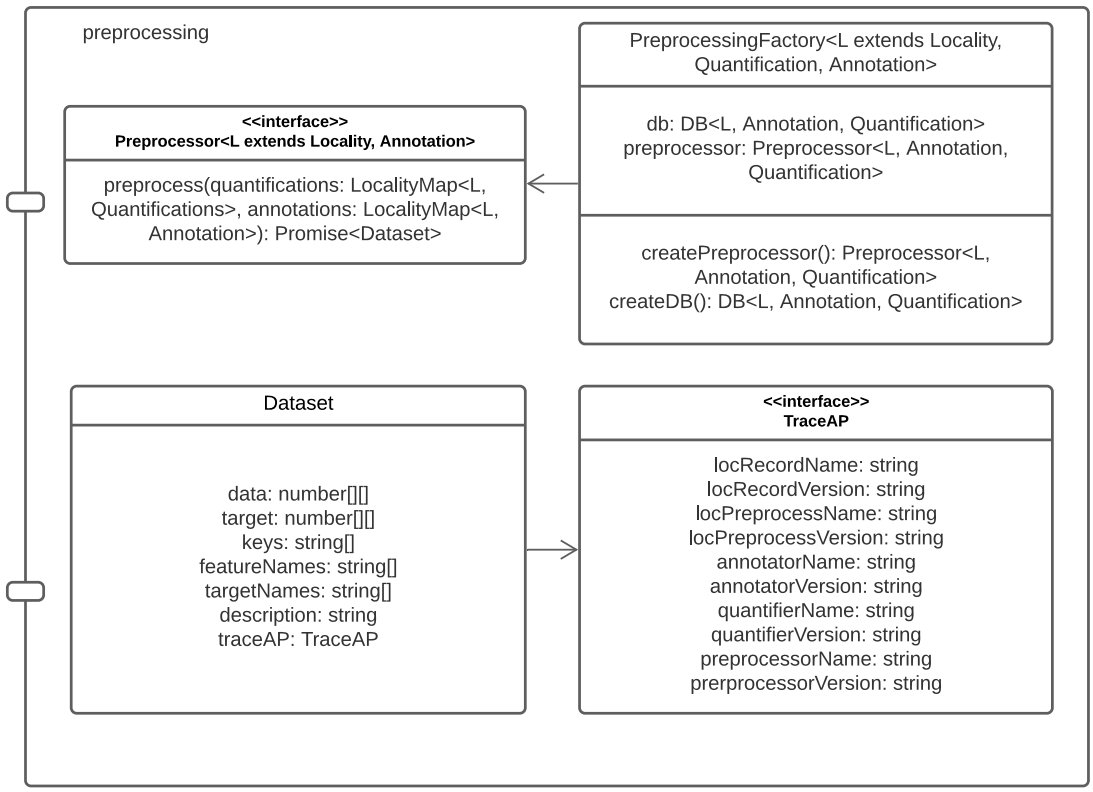

Abbildung 4.17: Framework: Preprocessing

Name	PREPROCESSING_TYPES	
Attribut	**Typ**	**Beispiel**
preprocessor	Preprocessor<CommitPath, number, SonarQubePredecessorMeasurement>	NullFilterPreprocessor
db	DB<CommitPath, number, SonarQubePredecessorMeasurement>	CommitPathsMongoDB
preprocessorFactory	PreprocessorFactory<CommitPath, number, SonarQubePredecessorMeasurement>	PreprocessorFactory

4.9.6 Pakete: Plugins

Die Komponenten des Frameworks wurden im Rahmen dieser Arbeit implementiert, um CommitPaths zu analysieren. Jedes der Pakete, das entwickelt wurde, ist als npm Paket in der npm-Registry registriert und lässt sich somit mittels `npm install` installieren. Jedes der Pakete ist öffentlich auf GitHub einsehbar. Alle einfachen

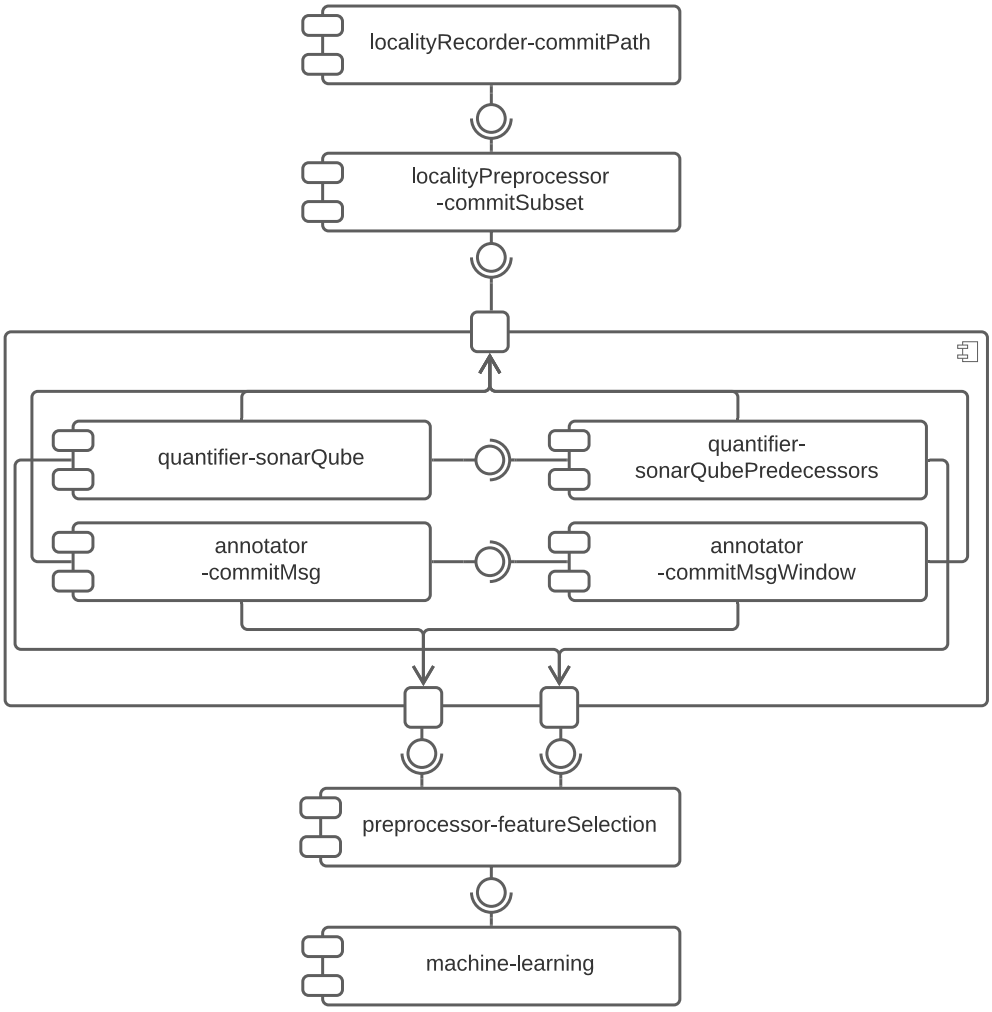

Abbildung 4.18: Komponentenüberblick

Pfeile der folgenden Diagramme bezeichnen, sofern nichts andere angegeben, einen Import.

LocalityRecorder

Es wurden `LocalityRecorder` für Commits und CommitPaths implementiert.

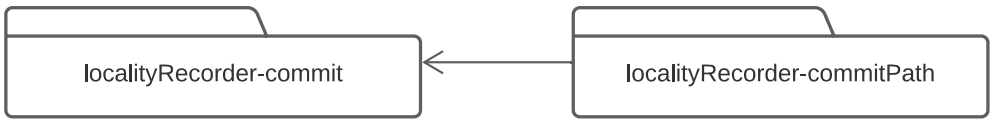

Abbildung 4.19: Pakete: LocalityRecorder

1. Commit

Das Paket LocalityRecorder-commit enthält den `CommitRecorder`, eine Implementierung des `LocalityRecorders` des bugfinder-frameworks, welcher Commits aus durch Git gemanagten Projekten lesen und parsen kann. Diese `Commits`-Objekte implementieren die `Locality`-Schnittstelle des bugfinder-frameworks.

Präfix	BUGFINDER_LOCALITYRECORDER_	
Name	COMMIT_TYPES	
Attribut	**Typ**	**Beispiel**
gitOptions	GitOptions	`{ baseDir: "../repositor" +"ies/TypeScript", maxConcurrentProcesses: 4 }`
git	Git	GitImpl
gitCommitParser	GitCommitParser	FormatParser
madFilesFrom CommitParser	MADFilesFrom Commit	MADFilesFromLog

2. CommitPath

Das Paket LocalityRecorder-commitPath enthält den `CommitPathRecorder`, eine Implementierung des `LocalityRecorders` des bugfinder-frameworks, welcher `CommitPath` -Objekte aus durch Git gemanagte Projekte erzeugt. Diese `CommitPaths`-Objekte implementieren die `Locality`-Schnittstelle des bugfinder-frameworks. Ein solches Objekt stellt einen Pfad in einem Commit dar. Für einen Commit kann es also mehrere `CommitPaths` geben.

Präfix	BUGFINDER_LOCALITYRECORDER_	
Name	COMMITPATH_TYPES	
Attribut	**Typ**	**Beispiel**
commitRecorder	LocalityRecorder<Commit>	CommitRecorder
commitToCommitPath Mapper	CommitToCommitPath Mapper	DefaultCommitPathMapper

DB

Es wurden DBs für Commits und CommitPaths, für die Nutzung von MongoDB, implementiert.

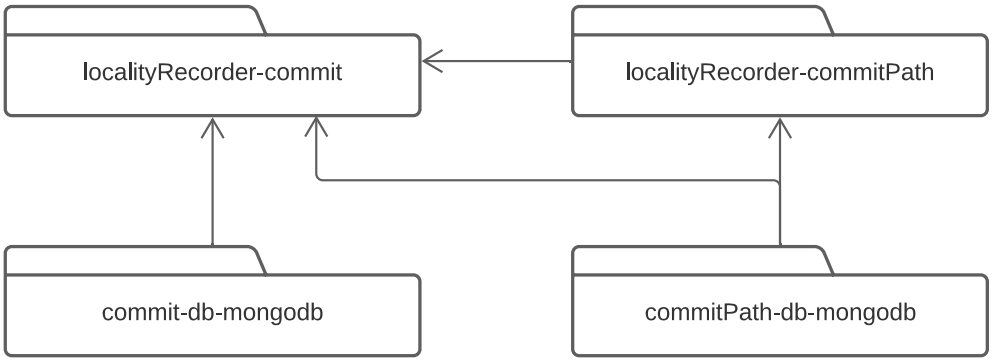

Abbildung 4.20: Pakete: Datenbanken

1. Commit

Das Paket enthält die `CommitsMongoDB`, welche die `DB`-Schnittstelle des Framework-Pakets implementiert.

Präfix	BUGFINDER_COMMIT_DB_	
Name	MONGODB_TYPES	
Attribut	**Typ**	**Beispiel**

mongoDBConfig	MongoDBConfig	`{ url: "mongodb://localhost" + ":27017", dbName: "TypeScript" }`

2. CommitPath

Das Paket enthält die `CommitPathsMongoDB`, welche die `DB`-Schnittstelle des Framework-Pakets implementiert.

Präfix	BUGFINDER_COMMITPATH_DB_	
Name	MONGODB_TYPES	
Attribut	**Typ**	**Beispiel**
mongoDBConfig	MongoDBConfig	`{ url: "mongodb://localhost" + ":27017", dbName: "TypeScript" }`

LocalityPreprocessor

Es wurde ein `LocalityPreprocessor` für `CommitPaths` implementiert, welcher CommitPaths einer Untermenge aller Commits filtert.

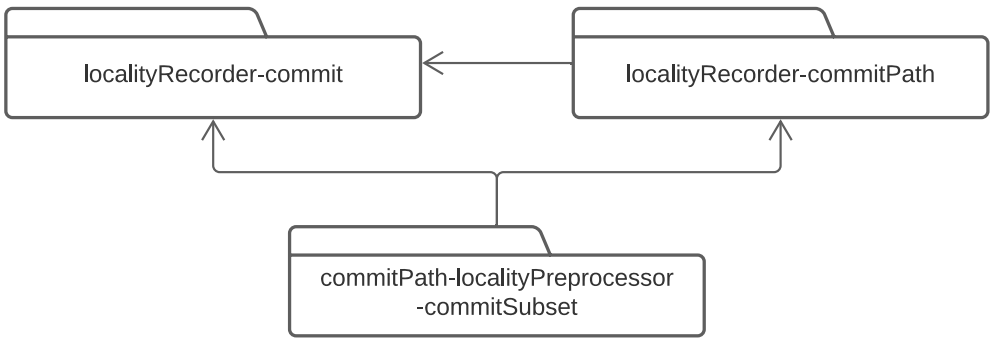

Abbildung 4.21: Pakete: LocalityPreprocessors

Diese Komponente kann konfiguriert werden hinsichtlich der skip Commits, die ausgelassen werden sollen; `n` Commits, die nach den Skip-Commits beibehalten werden sollen; und der pathsHandling-Konfiguration, welche angibt, was für Pfade je Commit injiziert werden sollen, falls sie injiziert werden sollen, ob Pfade auch injiziert werden sollen, wenn im Commit keine Pfade außer der injizierten übrig bleiben würden und den zu inkludieren und exkludieren Pfaden.

Präfix	BUGFINDER_COMMITPATH_LOCALITYPREPROCESSOR_	
Name	COMMITSUBSET_TYPES	
Attribut	**Typ**	**Beispiel**
skip	number	10000
n	number	5000
pathsHandling	PathsHandling	`{ injections: ["src"], injectOnEmptyPaths: false, pathIncludes: [/(.*\/)?src\/.*\.ts$/], pathExcludes: [/(.*\/)?src\/.*\.d\.ts$/, /(.*\/)?unittests\/.*\.ts/] }`

Annotator

Es wurden mehrere `Annotator` für CommitPaths implementiert.

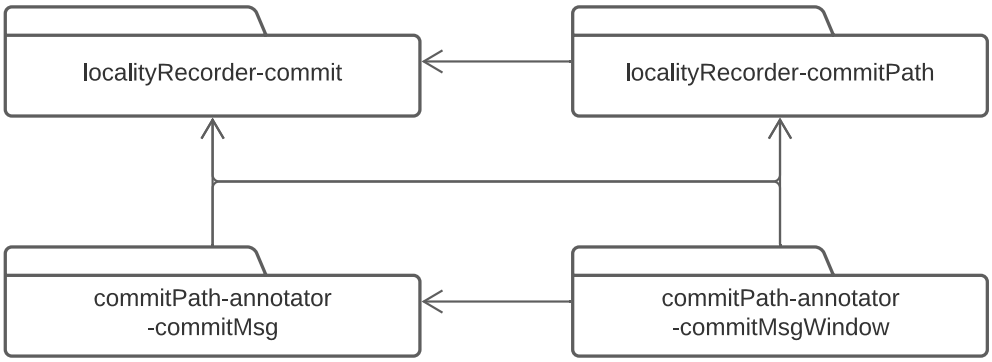

Abbildung 4.22: Pakete: Annotators

1. CommitMsgAnnotator

Das Paket CommitMsgAnnotator enthält den `CommitPathsAnnotator`, welcher die Annotator-Schnittstelle des Framework-Pakets implementiert. Dieser annotiert einen CommitPath binär anhand des Commits. Betrachtet wird die Beschaffenheit der Commit-Message, die Anzahl der beeinflussten Dateien des Commits, die Art des Commits und das Vorkommen von Testdateien.

Präfix	BUGFINDER_COMMITPATH_	
Name	ANNOTATOR_COMMITMSG_TYPES	
Attribut	**Typ**	**Beispiel**
testFileMatcher	RegExp	`/(test?\/.*\.*)/`

2. CommitMsgWindow

Das Paket CommitMsgWindow enthält den `CommitPathsWindowAnnotator`, welcher die Annotator-Schnittstelle des Framework-Pakets implementiert. Dieser annotiert einen CommitPath anhand Summe der Annotationen eines Annotierers für CommitPaths. Konfiguriert werden kann die Anzahl `nPred`, der zu betrachtenden, vorangegangen Änderungen und die Anzahl `nPost`, der zu betrachtenden, nachfolgenden Änderungen einer Datei. Ebenfalls die Berücksichtigung des aktuellen CommitPaths, das einzigartige Vorkommen vorangegangener oder nachfolgender Änderungen (= CommitPaths) und das gültige Annotieren in den Fällen von weniger als `nPred` respektive `nPost` vorkommenden Änderungen in allen CommitPaths kann konfiguriert werden.

Eine mögliche Verwendung des `CommitPathsWindowAnnotator` ist die Annotations anhand der Summe der binären Annotationen, mittels des im 1. Punkt genannten Annotierers `CommitPathAnnotator`, mehrerer nachfolgender Änderungen.

Präfix	BUGFINDER_COMMITPATH_ANNOTATOR_	
Name	COMMITMSGWINDOW_TYPES	
Attribut	**Typ**	**Beispiel**
commitPathAnnotator	Annotator<CommitPath>	CommitPathAnnotator
nPre	number	0
nPost	number	5

useCurrent	boolean	false
upToN	boolean	false
uniqueModePre	boolean	true
uniqueModePost	boolean	true

Quantifier

Es wurden mehrere `Quantifier` für `CommitPath`s implementiert.

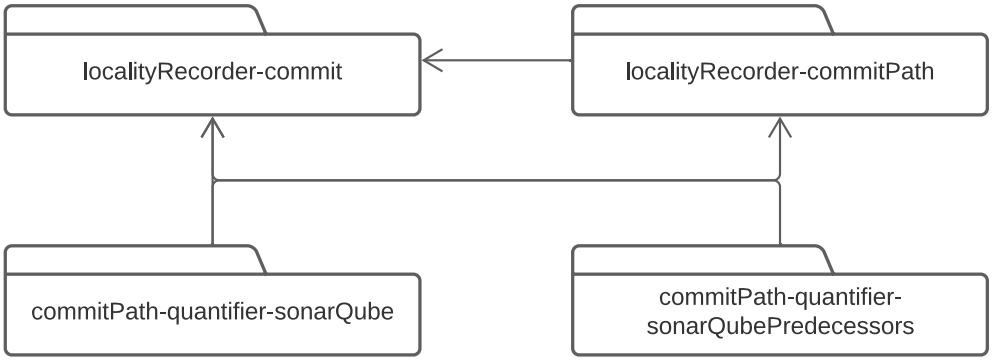

Abbildung 4.23: Pakete: Quantifiers

1. SonarQube

Das Paket SonarQube enthält den `SonarQubeQuantifier`, welcher die `Quantifier`-Schnittstelle des Framework-Pakets implementiert. Dieser `Quantifier` quantifiziert mithilfe von SonarQube `CommitPath`s. Es werden diverse Softwaremetriken gemessen. Unterstützt werden die SonarScanner.bat-kompatiblem (siehe offizielle SonarQube Dokumentation) Sprachen. Verwendet wurde SonarQube in der Version 9.0.1.46107.

Präfix	BUGFINDER_COMMITPATH_QUANTIFIER_	
Name	SONARQUBE_TYPES	
Attribut	**Typ**	**Beispiel**
sonarQubeConfig	SonarQubeConfig	`{ propertiesPath: "./typescript.properties", sonarQubeURL: "http://localhost:9000/", id: "admin", pw: "sonarqubepassword", preHooks: hooks }`
git	Git	GitImpl

Voraussetzung der Verwendung dieses Quantifizierers ist die Installation von SonarQube, sowie das Laufen des SonarQube-Servers und der Existenz eines bereits konfigurierten Projekts im Webinterface, welcher zur Quantifizierung verwendet werden kann. Dieses Projekt muss in der properties-Datei angegeben werden. Der propertiesPath der `SonarQubeConfig` ist der Pfad zur SonarQube-Properties-Datei, welche Sie der offiziellen SonarQube-Dokumentation oder exemplarisch dem bugFinder-Paket entnehmen können. Möchten Sie die `GitImpl` des Pakets localityRecorder-commit injizieren, müssen Sie die Abhängigkeiten des localityRecorder-commit-Pakets injizieren. Sehen Sie hierfür die Dokumentation auf S. 75)

2. SonarQubePredecessors

Das Paket SonarQubePredecessors enthält den `SonarQubePredecessorQuantifier`, welcher die Quantifier-Schnittstelle des Framework-Pakets implementiert. Dieser `Quantifier` quantifiziert mithilfe des `SonarQubeQuantifier` CommitPaths. Auf Basis der Softwaremetriken für Vorgänger- und Nachfolger-CommitPaths werden für jede Softwaremetrik des `SonarQubeQuantifier` der Minimalwert, Maximalwert und der arithmetische Durchschnitt aller Metriken aller betrachteten CommitPaths, die minimale Differenz, maximale Differenz, die arithmetische Durchschnittsdifferenz, die minimale relative, maximale relative und arithmetisch durchschnittliche relative Differenz zweier aufeinander folgender CommitPaths berechnet. Unterstützt werden die SonarScanner.bat-kompatiblem (siehe offizielle SonarQube Dokumentation) Sprachen. Verwendet wurde SonarQube in der Version 9.0.1.46107.

Präfix	BUGFINDER_COMMITPATH_QUANTIFIER_	
Name	SONARQUBEPREDECESSORS_TYPES	
Attribut	**Typ**	**Beispiel**
sonarQube	SonarQubeQuantifier	SonarQubeQuantifier
db	DB<CommitPath, any, SonarQubeMeasurement>	CommitPathsMongoDB
cacheID	string	SonarQubeMeasurements
n	number	5
upToN	boolean	false
uniqueMode	boolean	false
useThisCommitPath	boolean	true
cache	Cache	RAMCache

Voraussetzung der Verwendung dieses Quantifizierers ist die Installation von SonarQube, sowie das Laufen des SonarQube-Servers und der Existenz eines bereits konfigurierten Projekts im Webinterface, welcher zur Quantifizierung verwendet werden kann. Dies ist nicht notwendig, sofern alle CommitPaths bereits mittels SonarQube quantifiziert wurden und diese in der Datenbank unter `cacheID` zu finden sind. Möchten Sie den SonarQubeQuantifier benutzen, müssen Sie dessen Dependencies injizieren. Sehen Sie hierfür die Dokumentation des vorherigen Quantifizierers auf S. 80. Möchten Sie CommitPathsMongoDB nutzen, müssen Sie dessen Dependencies injizieren. Sehen Sie hierfür die Dokumentation auf S. 77.

Preprocessor

Das Paket featureSelection enthält den NullFilterPreprocessor, welches den `Preprocessor` des bugFinder-Frameworks implementiert.

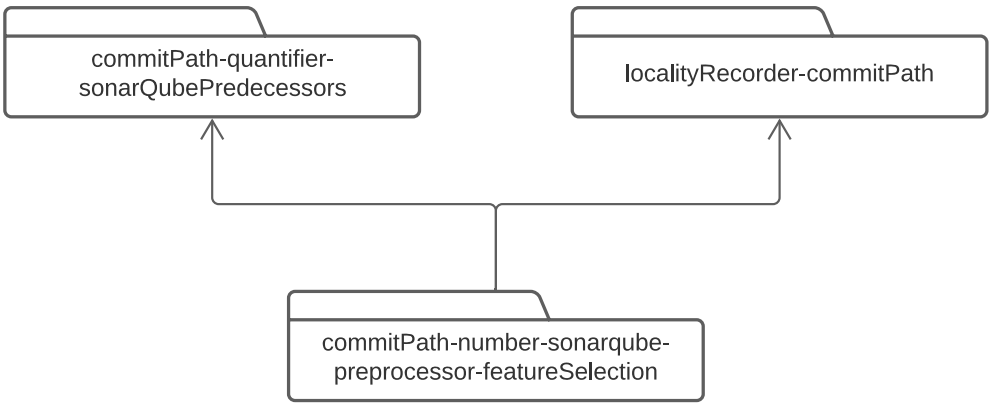

Abbildung 4.24: Paket: Preprocessor

Injiziert wird eine Beschreibung (`description`), welche den Datensatz und die Strategie zur Erzeugung des Datensatzes beschreibt. Ebenfalls werden Trace-Daten gespeichert, welche Informationen zu den verwendeten Plugins und derer Versionen enthalten.

Präfix	BUGFINDER_NUMBER_SONARQUBE_PREPROCESSOR_	
Name	FEATURESELECTION_TYPES	
Attribut	**Typ**	**Beispiel**
traceAP	TraceAP	`{ annotatorName: "bugfinder-commitpath-annotator-commitmsgwindow", annotatorVersion: "1.0.1", locPreprocessName: "bugfinder-commitpath-localitypreprocessor-commitsubset", locPreprocessVersion: "1.12.0", locRecordName: "bugfinder-localityrecorder-commitpath", locRecordVersion: "1.23.4", preprocessorName: "bugfinder-commitpath-number-sonarqubepredecessor measurement-preprocessor-selectiontransformer", preprocessorVersion: "1.0.1", quantifierName: "bugfinder-commitpath-quantifier-sonarqubepredecessors", quantifierVersion: "1.1.8" }`
description	string	"This dataset contains samples of code metrics of files and a target that indicated the number of fixes that occured to that file. SonarQube was used to measure atomic measurements. After that measurements of 3 predecessor CommitPaths were taken into account to calculate SonarQubePredecessorMeasurements. Predecessors CommitPaths are the last changes of file in predecessors commits. The annotations are the number of fix indicating commit messages in the 3 next CommitPaths (next changes to that file). A fix indicating commit msg is a msg that contains words like bug, fix, error or fail, affects less or equal than 2 files is not a merge commit and if the CommitPath is not a test file."

ignoreFeatures	string[]	<pre>1 [
		2 "minValClasses",
		3 "minValFunctions",
		4 "minValCoverage"
		5]</pre>

ignorePaths	string	[/src$/]

4.10 Entwurfsentscheidungen

Es wurden verschiedene Entwurfsentscheidungen getroffen, welche die Architektur und Vorgehensweise dieses Projektes prägen.

4.10.1 Kardinalitäten

Das Quantifizieren, sowie auch das Annotieren, erzeugen je Lokalität genau ein Quantifizierungs- bzw. Annotations-Objekt. Dieses Vorgehen soll sicherstellen, dass eine konstante Dimensionenzahl nach dem Preprocessing je Lokalität erzeugt wird. In Zukunft soll das die Entwicklung einer generischen Preprocessing-Komponente ermöglichen, welche lediglich das Zusammenfügen von Quantifizierungs- und Annotations-Objekten zu einem Datensatz ermöglicht.

4.10.2 Austauschbarkeit der Datenbankschnittstelle

Die Datenbankschnittstelle wurde austauschbar gestaltet, sodass in Zukunft andere Datenbanktypen unterstützt werden können. Dies soll eine etwaige Anbindung an andere, unter Umständen bereits bestehende, Machine-Learning-Pipelines von Machine-Learning-Experten vereinfachen.

4.10.3 Machine Learning als Template

Das Machine Learning wurde lediglich als Template und nicht als Plugin erstellt. Das hat den Hintergrund, dass das maschinelle Lernen selbst experimentell ist und die Komponenten des maschinellen Lernens bereits in modularisierter und parametrierbarer Form in Scitkit-Learn, aber auch anderen Bibliotheken, vorliegen. Das Teilen der Ergebnisse des maschinellen Lernens, insbesondere der Algorithmen, welche für diese Arbeit zum Einsatz kommen, ist jedoch durchaus wichtig.

4.10.4 Verwendung von SonarQube

Es wurde SonarQube zum Quantifizieren verwendet, da dieses Tool viele Sprachen unterstützen kann und damit aufzeigt, dass Projekte in verschiedenen Sprachen durch diese Arbeit analysiert werden könnten. SonarQube unterstützt in der Community Edition 17 Sprachen, ist kostenfrei und Open-Source (Stand 29.11.2021). In bezahlbaren Versionen werden mehr Sprachen unterstützt.

4.11 Qualitätsszenarien

Jenseits der bereits genannten qualitativen Ziele gibt es noch einige weitere relevante Qualitätsszenarien. Diverse Ziele, welche an diese Arbeit gestellt wurden und welche, die für die Weiterentwicklung dieser Arbeit vermutet werden. Nach ISO/IEC 25010 sind mit Ausnahme der Dimension Sicherheit alle weiteren Softwarequalitäten relevant. Es wurde keine repräsentative Befragung der Stakeholder durchgeführt, die folgenden Szenarien beruhen auf Annahmen über die Interessen, sowie auf Annahmen zu den verfügbaren Ressourcen der Stakeholder.

4.11.1 Zeitverhalten

Qualitätsszenarien für das Zeitverhalten unterscheiden sich insbesondere für die Lern- und die Prädiktionsphase.

Lernphase

Forscher, welche im Rahmen ihrer Abschlussarbeit diese Arbeit weiterentwickeln könnten, sind in ihrer Bearbeitungszeit beschränkt. Deshalb sollten ausreichende Ergebnisse in einem Teil der Bearbeitungszeit berechnet werden können.

Lernphase: Messen von Lokalitäten

Ursprung	Abschlussarbeiter: Forscher
Stimulus	Misst Lokalitäten.
Umgebung	Laufzeit, Lernphase
Artefakt	LocalityRecording-Komponente und Realisierungen der LocalityRecorder-Komponente.
Antwort	Es werden Lokalitäten gemessen.
Antwort-Messung	Das Messen von je 100.000 Lokalitäten erfolgt in unter einem Tag.

Lernphase: Vorverarbeiten von Lokalitäten

Ursprung	Abschlussarbeiter: Forscher
Stimulus	Verarbeitet Lokalitäten vor.
Umgebung	Laufzeit, Lernphase
Artefakt	LocalityPreprocessing-Komponente und Realisierungen der LocalitPreprocessor-Komponente.
Antwort	Es werden Lokalitäten aus einer Datenbank gelesen, vorverarbeitet und die vorverarbeiteten Lokalitäten in einer Datenbank abgelegt.
Antwort-Messung	Die Antwort erfolgt innerhalb von 30 Minuten.

Lernphase: Quantifizieren von Lokalitäten

Ursprung	Abschlussarbeiter: Forscher
Stimulus	Quantifiziert Lokalitäten.
Umgebung	Laufzeit, Lernphase
Artefakt	Quantifying-Komponente und Realisierungen der Quantifier-Komponente.
Antwort	100.000 Lokalitäten werden aus einer Datenbank gelesen und quantifiziert. Ergebnisse werden in einer Datenbank persistiert.
Antwort-Messung	Die Antwort erfolgt in unter drei Monaten je 100.000 Zeilen Code des zu quantifizierenden Projekts.

Lernphase: Annotieren von Lokalitäten

Ursprung	Abschlussarbeiter: Forscher
Stimulus	Annotiert Lokalitäten.

Umgebung	Laufzeit, Lernphase
Artefakt	Annotating-Komponente und Realisierungen der Annotator-Komponente.
Antwort	100.000 Lokalitäten werden aus einer Datenbank gelesen und annotiert. Ergebnisse werden in einer Datenbank persistiert.
Antwort-Messung	Die Antwort erfolgt in unter 15 Tagen.

Lernphase: Datenvorverarbeitung

Ursprung	Abschlussarbeiter: Forscher
Stimulus	Führt die Vorverarbeitung der gemessenen Daten aus.
Umgebung	Laufzeit, Lernphase
Artefakt	Preprocessing-Komponente und Realisierungen der Preprocessor- und Datenbank-Komponente.
Antwort	Verarbeitet Daten von 100.000 annotierten und quantifizierten Lokalitäten vor. Ergebnisse werden in einer Datenbank persistiert.
Antwort-Messung	Die Antwort benötigt maximal eine Stunde.

Prädiktionsphase

Anwender nutzen diese Arbeit, um Fehler zu identifizieren (Prädiktionsphase, siehe Abb. 4.7, S. 61). Ein Anwendungsszenario ist die Verwendung nach jedem Entwicklungsschritt. Eine zügige Untersuchung auf Fehler ist daher von Interesse. Daraus resultieren Anforderungen an das Zeitverhalten der einzelnen Komponenten, welche die einzelnen Schritte der Prädiktionsphase realisieren. Ein explizites Zeitverhalten der einzelnen Schritte lässt sich schlecht generisch verschriftlichen. Im Folgenden werden daher exemplarisch vermutete Qualitätsszenarien an die in dieser Arbeit realisierten Plugins aufgezeigt.

Prädiktionsphase: Untersuchung von Lokalitäten auf Fehler

Ursprung	Anwender
Stimulus	Untersucht Lokalitäten.
Umgebung	Laufzeit, Prädiktionsphase
Artefakt	Alle Realisierungen der Knowledge-Source- und Control-Komponenten, ausgenommen der Annotating- und Annotator-Realisierungen. Die Datenbankrealisierung.
Antwort	Es werden Lokalitäten auf Fehler untersucht.
Antwort-Messung	Die Untersuchung von zehn Lokalitäten dauert maximal zwei Minuten.

Prädiktionsphase: Messen von Lokalitäten

Ursprung	Anwender
Stimulus	Untersucht Lokalitäten auf Fehler.
Umgebung	Laufzeit, Prädiktionsphase
Artefakt	LocalityRecording-Komponente und Plugin zur Realisierung der LocalityRecorder-Komponente für CommitPaths.
Antwort	Es werden CommitPaths der vorangegangenen Änderungen der CommitPaths der letzten Commits gemessen.
Antwort-Messung	Das Messen der letzten bis zu zehn vorangegangenen Änderungen der letzten bis zu zehn CommitPaths dauert maximal fünf Sekunden.

Prädiktionsphase: Vorverarbeiten von Lokalitäten

Ursprung	Anwender
Stimulus	Untersucht Lokalitäten auf Fehler. Lokalitäten wurden gemessen.
Umgebung	Laufzeit, Prädiktionsphase
Artefakt	LocalityPreprocessing-Komponente und Plugin zur Realisierung der LocalityPreprocessor-Komponente für CommitPaths.
Antwort	Es werden CommitPaths vorverarbeitet.
Antwort-Messung	Das Vorverarbeiten von bis zu 100 CommitPaths dauert maximal fünf Sekunden.

Prädiktionsphase: Quantifizieren von Lokalitäten

Ursprung	Anwender
Stimulus	Untersucht Lokalitäten auf Fehler. Lokalitäten wurden vorverarbeitet.
Umgebung	Laufzeit, Prädiktionsphase
Artefakt	Quantifying-Komponente und Plugin zur Realisierung der Quantifier-Komponente für CommitPaths.
Antwort	Es werden CommitPaths quantifiziert.
Antwort-Messung	Das Quantifizieren von bis zu 100 CommitPaths dauert maximal 1 Minute und 40 Sekunden.

Prädiktionsphase: Vorverarbeiten quantifizierter Lokalitäten

Ursprung	Anwender
Stimulus	Untersucht Lokalitäten auf Fehler. Lokalitäten wurden quantifiziert.
Umgebung	Laufzeit, Prädiktionsphase

Artefakt	Preprocessing-Komponente und Plugin zur Realisierung der preprocess-Aktivität der Prädiktionsphase.
Antwort	Es werden quantifizierte Lokalitäten in ein Format gebracht, welches für das Modell der Prädiktionsaktivität geeignet ist.
Antwort-Messung	Die Antwort benötigt maximal fünf Sekunden.

Prädiktionsphase: Prädiktion

Ursprung	Anwender
Stimulus	Untersucht Lokalitäten auf Fehler. Quantifizierte Lokalitäten wurden vorverarbeitet.
Umgebung	Laufzeit, Prädiktionsphase
Artefakt	Komponenten zur Realisierung der Prädiktionsaktivität.
Antwort	Es werden Prädiktionen für die Daten aus den vorverarbeiteten quantifizierten Lokalitäten getroffen.
Antwort-Messung	Die Antwort benötigt maximal fünf Sekunden.

4.11.2 Ressourcen-Nutzung

Insbesondere Abschlussarbeiter können in ihren Ressourcen eingeschränkt sein. Idealerweise sollten alle Daten in den Arbeitsspeicher eines typischen Computers eines Abschlussarbeiters passen.

Messen von Lokalitäten

Ursprung	Abschlussarbeiter: Forscher
Stimulus	Misst CommitPaths.
Umgebung	Laufzeit

Artefakt	LocalityRecording-Komponente und Plugins aus Kapitel 4.9: LocalityRecorder- und Datenbank-Implementierungen.
Antwort	Es werden als CommitPaths gemessen und in einer Datenbank persistiert.
Antwort-Messung	Der Speicherbedarf der CommitPaths beträgt maximal zwei GB.

Vorverarbeiten von Lokalitäten

Ursprung	Abschlussarbeiter: Forscher
Stimulus	Verarbeitet CommitPaths vor.
Umgebung	Laufzeit
Artefakt	LocalityPreprocessing-Komponente und Plugins aus Kapitel 4.9: LocalityPreprocessor- und Datenbank-Implementierungen.
Antwort	Es werden CommitPaths aus einer Datenbank gelesen, vorverarbeitet und die Ergebnisse der Vorverarbeitung in einer Datenbank abgelegt.
Antwort-Messung	Der Speicherbedarf der vorverarbeiteten CommitPaths beträgt maximal zwei GB.

Quantifizieren von Lokalitäten

Ursprung	Abschlussarbeiter: Forscher
Stimulus	Quantifiziert Lokalitäten.
Umgebung	Laufzeit
Artefakt	Plugins aus Kapitel 4.9: Quantifier- und Datenbank-Implementierungen.
Antwort	Die CommitPaths werden aus der Datenbank gelesen, quantifiziert und alle quantifizierten CommitPaths in einer Datenbank persistiert.

Antwort-Messung	Der Speicherbedarf der quantifizierten CommitPaths beträgt maximal zehn GB. Da die quantifizierten CommitPaths auch die CommitPaths enthalten, welche maximal zwei GB groß sein dürfen, dürfen die Quantifizierungen selbst maximal acht GB Speicher benötigen.

Annotieren von Lokalitäten

Ursprung	Abschlussarbeiter: Forscher
Stimulus	Annotiert Lokalitäten.
Umgebung	Laufzeit
Artefakt	Annotating-Komponente und Plugins aus Kapitel 4.9: Annotator- und Datenbank-Implementierungen.
Antwort	Die CommitPaths werden aus der Datenbank gelesen, annotiert und alle annotierten CommitPaths in einer Datenbank persistiert.
Antwort-Messung	Der Speicherbedarf der annotierten CommitPaths beträgt maximal vier GB. Da die annotierten CommitPaths auch die CommitPaths enthalten, welche maximal zwei GB groß sein dürfen, dürfen die Annotation selbst maximal zwei GB Speicher benötigen.

4.11.3 Nutzbarkeit

Die Nutzbarkeit (Usability) des Projekts ist eine wichtige Qualität, um andere Forscher anzuregen dieses Projekt zu nutzen.

Messen von Lokalitäten

Ursprung	Forscher
Stimulus	Lernt wie Lokalitäten gemessen werden.
Umgebung	Design-Zeit
Artefakt	Dokumentation der LocalityRecording- und LocalityRecorder-Komponente, sowie aller Plugins, welche die LocalityRecorder-Komponente realisieren.

Antwort	Die Dokumentation der betroffenen Artefakte, sowie die Architekturdokumentation des Projekts werden durchgelesen.
Antwort-Messung	Der Forscher ist in der Lage innerhalb eines Tages Lokalitäten zu messen und in einer Datenbank zu persistieren.

Vorverarbeiten von Lokalitäten

Ursprung	Forscher
Stimulus	Lern wie Lokalitäten gemessen werden.
Umgebung	Design-Zeit
Artefakt	Dokumentation der LocalityPreprocessing- und LocalityPreprocessor-Komponente, sowie aller Plugins, welche die LocalityPreprocessor-Komponente realisieren.
Antwort	Die Dokumentation der betroffenen Artefakte, sowie die Architekturdokumentation des Projekts werden durchgelesen.
Antwort-Messung	Der Forscher, welcher bereits das Messen von Lokalitäten gelernt hat, ist in der Lage innerhalb von vier Stunden Lokalitäten aus einer Datenbank zu lesen, vorzuverarbeiten und die vorverarbeiteten Lokalitäten in einer Datenbank zu persistieren.

Quantifizieren von Lokalitäten

Ursprung	Forscher
Stimulus	Lern wie Lokalitäten quantifiziert werden.
Umgebung	Design-Zeit
Artefakt	Dokumentation der Quantifying- und Quantifier-Komponente, sowie aller Plugins, welche die Quantifier-Komponente realisieren.
Antwort	Die Dokumentation der betroffenen Artefakte, sowie die Architekturdokumentation des Projekts werden durchgelesen.

Antwort-Messung	Der Forscher, welcher bereits das Vorverarbeiten von Lokalitäten gelernt hat, ist in der Lage innerhalb von vier Stunden Lokalitäten aus der Datenbank zu lesen, diese zu quantifizieren und die quantifizierten Lokalitäten wieder in der Datenbank abzulegen.

Annotieren von Lokalitäten

Ursprung	Forscher
Stimulus	Lernt wie Lokalitäten annotiert werden.
Umgebung	Design-Zeit
Artefakt	Dokumentation der Annotating- und Annotator-Komponente, sowie aller Plugins, welche die Annotator-Komponente realisieren.
Antwort	Die Dokumentation der betroffenen Artefakte, sowie die Architekturdokumentation des Projekts werden durchgelesen.
Antwort-Messung	Der Forscher, welcher bereits das Vorverarbeiten von Lokalitäten gelernt hat, ist in der Lage innerhalb von vier Stunden Lokalitäten aus der Datenbank zu lesen, diese zu annotieren und die annotierten Lokalitäten wieder in der Datenbank abzulegen.

Datenvorverarbeitung

Ursprung	Forscher
Stimulus	Lern wie annotierte und quantifizierte Lokalitäten vorverarbeitet werden.
Umgebung	Design-Zeit
Artefakt	Dokumentation der Preprocessing- und Preprocessor-Komponente, sowie aller Plugins, welche die Preprocessor-Komponente realisieren.
Antwort	Die Dokumentation der betroffenen Artefakte, sowie die Architekturdokumentation des Projekts werden durchgelesen.

Antwort-Messung	Der Forscher, welcher bereits das Annotieren und Quantifizieren von Lokalitäten gelernt hat, ist in der Lage innerhalb von vier Stunden quantifizierte und annotierte Lokalitäten aus der Datenbank zu lesen, diese vorzuverarbeiten und die Ergebnisse der Vorverarbeitung in einer Datenbank zu speichern.

Maschinelles Lernen

Ursprung	Forscher
Stimulus	Lernt die Verwendung der Machine-Learning-Komponente
Umgebung	Design-Zeit
Artefakt	Machine-Learning-Komponente-, Scikit-Learn-Dokumentation, sowie das Template für das maschinelle Lernen.
Antwort	Die Dokumentation der betroffenen Artefakte, die Scikit-Learn-Dokumentation der verwendeten Funktionen des Machine-Learning-Templates, sowie die Architekturdokumentation des Projekts werden durchgelesen.
Antwort-Messung	Der Forscher ist in der Lage ohne Vorkenntnisse das Template für das maschinelle Lernen innerhalb von drei Tagen zu nutzen, um ein Modell zu trainieren und dieses nutzen zu können.

Austauschen von Komponenten

Ursprung	Forscher
Stimulus	Lernt wie der Austausch von Komponenten erfolgt
Umgebung	Design-Zeit
Artefakt	Dokumentation der LocalityRecording-, LocalityRecorder-, LocalityPreprocessing-, LocalityPreprocessor-, Quantifying-, Quantifier-, Annotating-, Annotator-, Preprocessing- und Preprocessor-Komponente.
Antwort	Die Dokumentation der betroffenen Artefakte sowie die Architekturdokumentation des Projekts werden durchgelesen.

Antwort-Messung	Der Forscher ist in der Lage, die Komponenten LocalityRecorder, LocalityPreprocessor, Quantifier, Annotator und Preprocessor durch andere geeignete Komponenten auszutauschen.

Auffindbarkeit der Dokumentation

Ursprung	Forscher
Stimulus	Sucht Dokumentation einzelner Komponenten
Umgebung	Design-Zeit
Artefakt	Github-Repositories aller Komponenten dieses Projekts, diese Dokumentation.
Antwort	Die öffentlich zugängliche Github-Repositories werden eingesehen.
Antwort-Messung	Die Dokumentation der passenden Komponenten kann durch Lesen der README-Dateien aufgefunden werden.

4.11.4 Verlässlichkeit

Die Komponenten des Frameworks und deren Realisierungen sollten, ohne abzustürzen, ausreichende Datenmengen generieren.

Messen von Lokalitäten

Ursprung	Forscher
Stimulus	Misst Lokalitäten.
Umgebung	Laufzeit
Artefakt	LocalityRecording- und LocalityRecorder-Komponente, sowie alle Plugins welche die LocalityRecorder-Komponente realisieren.
Antwort	Nutzt die LocalityRecording-Komponente, um mit einer LocalityRecorder-Komponente Lokalitäten zu messen.

Antwort-Messung	LocalityRecording-Komponente misst, ohne abzustürzen, 250.000 Lokalitäten.

Vorverarbeiten von Lokalitäten

Ursprung	Forscher
Stimulus	Verarbeitet Lokalitäten vor.
Umgebung	Laufzeit
Artefakt	Preprocessing- und Preprocessor-Komponente, sowie alle Plugins welche die Preprocessor-Komponente realisieren.
Antwort	Nutzt die Preprocessing-Komponente, um mit einer Preprocessor-Komponente Lokalitäten vorzuverarbeiten.
Antwort-Messung	Preprocessing-Komponente verarbeitet, ohne abzustürzen, 250.000 Lokalitäten vor.

Quantifizieren von Lokalitäten

Ursprung	Forscher
Stimulus	Quantifiziert Lokalitäten.
Umgebung	Laufzeit
Artefakt	Quantifying- und Quantifier-Komponente, sowie alle Plugins welche die Quantifier-Komponente realisieren.
Antwort	Nutzt die Quantifying-Komponente, um mit einer Quantifier-Komponente Lokalitäten zu quantifizieren.
Antwort-Messung	Quantifying-Komponente quantifiziert, ohne abzustürzen, 100.000 Lokalitäten.

Annotieren von Lokalitäten

Ursprung	Forscher
Stimulus	Annotiert Lokalitäten.
Umgebung	Laufzeit
Artefakt	Annotating- und Annotator-Komponente, sowie alle Plugins welche die Annotator-Komponente realisieren.
Antwort	Nutzt die Annotating-Komponente, um mit einer Annotator-Komponente Lokalitäten zu annotieren.
Antwort-Messung	Annotating-Komponente annotiert, ohne abzustürzen, 100.000 Lokalitäten.

Datenvorverarbeitung

Ursprung	Forscher
Stimulus	Führt die Vorverarbeitung der gemessenen Daten aus.
Umgebung	Laufzeit
Artefakt	Preprocessing- und Preprocessor-Komponente, sowie alle Plugins welche die Preprocessor-Komponente realisieren.
Antwort	Nutzt die Preprocessing-Komponente, um mit einer Preprocessor-Komponente quantifizierte und annotierte Lokalitäten vorzuverarbeiten.
Antwort-Messung	Preprocessing-Komponente verarbeitet 100.000 quantifizierte und annotierte Lokalitäten, ohne abzustürzen, vor.

Prädiktionsphase

Ursprung	Anwender
Stimulus	Untersucht Lokalitäten.
Umgebung	Laufzeit, Prädiktionsphase

Artefakt	Die LocalityRecording-, LocalityPreprocessing-, Quantifying-, Preprocessing- und Prediction-Komponenten, sowie Realisierungen der LocalityRecorder-, LocalityPreprocesser-, Quantifier-, Preprocessor- und Predictor-Komponenten.
Antwort	Prädiziert Fehler in Lokalitäten.
Antwort-Messung	Die Prädiktion von 100 Lokalitäten erfolgt, ohne dass das Programm abstürzt.

4.11.5 Wartbarkeit

Ursprung	Forscher
Stimulus	Tauscht einer der Komponenten LocalityRecorder, LocalityPreprocessor, Quantifier, Annotator oder Preprocessor aus.
Umgebung	Design-Zeit
Artefakt	LocalityRecording-, LocalityRecorder, LocaltiyPreprocessing-, LocalityPreprocessor-, Quantifying-, Quantifier-, Annotating-, Annotator-, Preprocessing- und Preprocessor-Komponente.
Antwort	Konfiguration der Dependency-Injection der LocalityRecording-Komponente wird angepasst, sodass die neue LocalityRecorder-Komponente verwendet wird. Analog für die anderen Komponenten.
Antwort-Messung	Austausch einer Komponente erfolgt innerhalb von maximal 30 Minuten.

4.12 Risiken und technische Schulden

1. Die Arc42-Dokumentation ist nur auf Deutsch vorhanden. Das schränkt die Menge der möglichen Stakeholder drastisch ein.
2. Die Softwarequalität Verlässlichkeit ist unzureichend adressiert. Systemtests, die Verwendung eines Loggers, zur Handhabung von Ausnahmezuständen, sowie die Handhabung etlicher Ausnahmezustände wurden praktiziert, dennoch ist das Projekt unzureichend, insbesondere unzureichend auf alle

Qualitätsszenarien, getestet.

3. Es wurde unzureichend auf qualitative Laufzeitszenarien für die Prädiktionsphase eingegangen. Weitere Optimierungen werden benötigt.
4. Das Plugin LocalityRecorder-CommitPath verwendet für statische Funktionen den aus den SHARED_TYPES injizierten Logger nicht, da InversifyJS keine statischen Injektionen unterstützt.
5. Die Datenbankschnittstelle verletzt das Single-Responsibility-Principle und erfüllt damit nicht das Interface-Segregation-Principle. Ursprünglich gedacht, um die Erlernbarkeit des Frameworks zu begünstigen, sollte eine Aufteilung der Schnittstelle und die Implementierung aller Datenbank-Schnittstellen in einer Klasse (ggf. Wrapper-Klasse) die Erlernbarkeit nur geringfügig verschlechtern, aber die Modularität verbessern.
6. Einige Klassenbezeichnungen sind suboptimal gewählt worden. Die Klasse CommitPathsAnnotator suggeriert beispielsweise der einzige Annotator für CommitPaths zu sein.

4.13 Evaluation und Diskussion

Das Framework wurde genutzt, um das Open-Source-Projekt TypeScript zu untersuchen. Zur Realisierung der Blackboard wurde eine MongoDB-Unterstützung und Control-Komponenten für die Realisierung der einzelnen Pipeline-Schritte, entwickelt. Es wurden für alle Knowledge-Source-Komponenten der Architektur Implementierungen vorgenommen. Zur Messung der Lokalitäten wurde eine LocalityRecorder-Komponente entwickelt, mit welcher 295.428 CommitPaths als Lokalitäten gemessen wurden. Ein CommitPath bezeichnet eine Datei in einem Commit. Es wurde eine LocalityPreprocessor-Komponente entwickelt, welche aus allen Lokalitäten 17.250 CommitPaths, aus 10.000 Commits des TypeScript-Projekts, generierte. Als Quantifier-Komponente wurde eine Implementierung zur Unterstützung von SonarQube entwickelt. Diese berechnete für alle 17.250 Lokalitäten Softwaremetriken. Unter Verwendung der gemessenen Softwaremetriken, nutzte eine weitere Quantifier-Realisierung die Blackboard als Memoizations-Cache, um Aggregate der Softwaremetriken der letzten Änderungen einer Datei zu errechnen. Es wurden zwei Plugins entwickelt, um die Annotator-Komponente zu realisieren. Eins weist binär zu, ob ein CommitPath Fehlerbehebungs-indizierend ist. Das zweite Plugin berechnet die Summe der Ergebnisse des Ersteren über die nächsten CommitPaths. Die Ergebnisse der Quantifizierungen über die letzten Änderungen einer Datei, sowie die Annotationen über die nächsten Änderungen einer Datei, wurden mithilfe einer Preprocessor-Realisierung zu einem Datensatz zusammengefügt. Ein Machine-Learning-Template wurde entwickelt, welches diverse Modelle anhand eines Datensatzes trainiert und evaluiert. Die Modelle

werden persistiert.

4.13.1 Funktionale Anforderungen und qualitative Ziele

Es wurden alle funktionalen Anforderungen erfüllt. Die Anforderungen, die das Framework ermöglichen soll, wurden durch die konkreten Realisierungen aufgezeigt. Die qualitativen Ziele Austauschbarkeit und Installierbarkeit der Knowledge-Source-Komponenten wurde durch Dependency Injection und die Entwicklung von Paketen, welche in der npm-Registry registriert sind, realisiert. Da alle Schnittstellen des Frameworks generisch gehalten sind, kann der Lokalitätstyp ohne Anpassung weiterer Schnittstellen ausgetauscht werden.

4.13.2 Lizenzierung

Jegliche Software, die im Rahmen dieser Masterarbeit entwickelt wurde, wurde unter der ISC-Lizenz veröffentlicht. Sie ist besonders einfach gehalten, Open-Source geeignet und wenig restriktiv. Hierfür wurden alle installierten und verwendeten Pakete nach Lizenzen geprüft und restriktiv lizenzierte Pakete ausgetauscht und vermieden. Die SonarQube Community Edition fällt unter die GPLv3-Lizenz und wird mit dieser Arbeit nicht mit ausgeliefert. Die SonarQube-Software erfüllt die Kriterien der Free Software Foundation und kann daher ausgeführt, studiert, geändert und verbreitet werden. Sie besitzt eine Copyleft-Klausel, sodass abgeleitete Programme die Bedingungen der GPL-Lizenz erfüllen müssen. Die Verwendung der SonarQube Community Edition ist kostenfrei möglich, unter dieser werden 15 Programmiersprachen unterstützt. Versionen, welche weitere Programmiersprachen unterstützen, sind kostenpflichtig.[1]

4.13.3 Prädiktionsphase

Die Prädiktionsphase wurde in dieser Arbeit nicht vollständig und explizit modelliert. Es ist wahrscheinlich, dass ohne Anpassungen der Komponenten zur Realisierung der Lernphase, Qualitätsszenarien für die Prädiktionsphase nicht erfüllt werden können. Womöglich kann das Vorverarbeiten von Lokalitäten in der Prädiktionsphase ausgelassen werden, das Zwischenspeichern und erneute Laden zwischen allen Pipelineschritten ist in der Prädiktionsphase nicht notwendig. Inkrementelle Messungen der CommitPaths und atomaren Quantifizierungen einzelner CommitPaths werden zur Erfüllung der Anforderungen an die Laufzeit der Prädiktionsphase voraussichtlich unvermeidbar. Mit inkrementellen Messungen ist

[1] https://www.sonarsource.com/plans-and-pricing/ (Stand 28.11.2021)

gemeint jeden CommitPath während der Entwicklung genau ein mal zu messen und zu quantifizieren und die CommitPaths und atomaren Quantifizierungen in einer Datenbank abzulegen. Mit atomaren Quantifizierungen sind Messungen der Softwaremetriken einzelner CommitPaths mit SonarQube gemeint. Diese können verwendet werden, um in kurzer Zeit Aggregate über Softwaremetriken der letzten Änderungen eines CommitPaths zu berechnen. Die Messung von bis zu 100 CommitPaths mit SonarQube könnte die Messung von bis zu 100 verschiedenen Commits zur Folge haben, was mit aktuell üblichen Computern von Anwendern wesentlich länger als zwei Minuten dauert.

Die Preprocessing-Komponente ist ohne Weiteres nicht für die Prädiktionsphase geeignet, da Annotationen benötigt werden, um einen Datensatz zu generieren. Unter Verwendung von Mock-Annotationen ist es möglich, die Preprocessing-Komponente wiederzuverwenden. Die Prädiktion unter Verwendung der trainierten Modelle stellt kein Laufzeitrisiko dar. Untersuchungen und Optimierungen zur Prädiktionsphase werden zukünftigen Arbeiten überlassen.

4.13.4 Qualitätsszenarien

Viele Qualitätsszenarien wurden adressiert und werden erfüllt. Das Laufzeitverhalten des Plugins zur Realisierung der Quantifier-Komponente erfordert besondere Aufmerksamkeit. Die durchgeführten Quantifizierungen haben auf mehreren Rechnern parallel mehrere Wochen in Anspruch genommen. Durch Steigerung der Verlässlichkeit, primär durch Verwendung eines Loggers, aber auch durch einige Systemtests, sollten geeignete Datenmengen zügiger erzeugt werden können. Zur Erzeugung größerer Datenmengen, welche für dieses Projekt von Interesse sind, muss voraussichtlich eine Unterstützung für Cluster-Computing umgesetzt werden. Das Quantifizieren von 100.000 CommitPaths mit SonarQube innerhalb von 3 Monaten je 100.000 Zeilen Quellcode wird mit der aktuellen Implementierung, auf einem einzelnen typischen PC, nicht erfüllt.

Qualitätsszenarien, welche die Ressourcennutzung adressieren, konnten, durch Normalisierung der CommitPaths, umgesetzt werden. Alle CommitPaths des TypeScript Projekts nahmen ohne Normalisierung mehrere 100 GB in Anspruch, mit 63 MB. Die Quantifizierung von 17.250 CommitPaths mit SonarQube benötigt 61.1 MB. Die Qualitätsszenarien werden bei einem 100-fachen der Datenmenge weiterhin erfüllt.

Die Nutzbarkeit des Projektes wurde durch ISC-Lizenzierte Veröffentlichung auf Github und npm, die Verwendung von Jupyter Notebook inklusive Dokumentation einzelner Schritte, die Verwendung der gut dokumentierten Scikit-Learn-Bibliothek, sowie der Architekturdokumentation, begünstigt. Die Qualitätsszenarien zur Nutzbarkeit sind diskutabel. Deutlich strengere Anforderungen sind denkbar. Bedingt durch die Komplexität der Aufgabenstellung, der Einzelschritte, die zur Erfüllung

der funktionalen Anforderungen benötigt werden, sowie durch qualitative Ziele zur Austauschbarkeit, ist die Nutzbarkeit des Projekts beschränkt.

4.14 Glossar

Begriff	**Erklärung**
Softwarequalität: Zeitverhalten	„Grad, in welcher die Reaktions- und Verarbeitungszeiten sowie die Durchsatzraten eines Produkts oder Systems, bei der Ausführung seiner Funktionen, den Anforderungen entspricht.“ („Norm ISO/IEC 25010“, 2011, übersetzt)
Softwarequalität: Ressourcen-Nutzung	„Grad, in welcher die Menge und Art der Ressourcen, die ein Produkt oder System, bei der Ausführung seiner Funktionen, nutzt, den Anforderungen gerecht wird.“ („Norm ISO/IEC 25010“, 2011, übersetzt)
Softwarequalität: Nutzbarkeit (Usability)	„Grad, in dem ein Produkt oder System von spezifizierten Benutzern verwendet werden kann, um spezifizierte Ziele mit Effektivität, Effizienz und Zufriedenheit, in einem bestimmten Nutzungskontext, zu erreichen. “ („Norm ISO/IEC 25010“, 2011, übersetzt)
Softwarequalität: Verlässlichkeit	„Grad, in dem ein System, ein Produkt oder eine Komponente bestimmte Funktionen, unter bestimmten Bedingungen, für eine bestimmte Zeit, ausführt.“ („Norm ISO/IEC 25010“, 2011, übersetzt)
Softwarequalität: Wartbarkeit	„Grad der Effektivität und Effizienz, mit der ein Produkt oder System, von vorgesehenen Wartungspersonal, verändert werden kann.“ („Norm ISO/IEC 25010“, 2011, übersetzt)
Softwarequalität: Austauschbarkeit	„Grad, in dem ein Produkt ein anderes spezifiziertes Softwareprodukt, für denselben Zweck, in derselben Umgebung, ersetzen kann.“ („Norm ISO/IEC 25010“, 2011, übersetzt)
Softwarequalität: Installierbarkeit	„Grad der Effektivität und Effizienz, mit der ein Produkt oder System, in einer bestimmten Umgebung, erfolgreich installiert und/oder deinstalliert werden kann.“ („Norm ISO/IEC 25010“, 2011, übersetzt)
Lokalität	Eine Lokalität bezeichnet eine Stelle, über die Quantifizierungen und Annotationen abgeleitet werden können. Beispiele für Lokalitäten sind Commits, Pfade in Commits oder Aktienkurs-Webseiten-Zustände.

Annotierer (Annotator)	Ein Annotierer weist Lokalitäten Klassen oder Zahlenwerte zu. Beispiel für einen Annotierer für den Lokalitätstypen *Pfad in einem Commit*: Zuweisung der Anzahl der Fehler-Behebungs-Indizierenden Commit-Nachrichten, der nächsten fünf Commits, zu jedem Pfad in einem Commit.
Quantifizierer (Quantifier)	Ein Quantifizierer weist Lokalitäten metrische Werte (Quantifizierungen/Quantitäten) zu. Beispiel für einen Quantifizierer für den Lokalitätstypen *Pfad in einem Commit*: Messung der Softwaremetriken dieses Pfades mithilfe von SonarQube
CommitPath	Ein CommitPath bezeichnet die Lokalität *Pfad in einem Commit*. Dies entspricht einem Zustand einer Datei, während der Entwicklung, eines mit git-gemanagten Softwareprojektes.
Code Smell	Metaphorische Bezeichnung für Quellcode, für welchen Verbesserungen nahegelegt werden.
MongoDB	„MongoDB ist eine dokumentenorientierte Datenbank, die auf einfache Entwicklung und Skalierung ausgelegt ist.“ („MongoDB Dokumentation“, eingesehen am 15.11.2021, frei übersetzt)
Scikit-Learn	„Scikit-learn ist ein Python-Modul, das eine breite Palette modernster Algorithmen des maschinellen Lernens, für mittelgroße überwachte und unüberwachte Probleme, integriert.“ (Pedregosa et al., 2011, übersetzt)
Sample	Ein Sample bezeichnet einen Datenpunkt. Im Kontext dieser Arbeit werden Daten je Lokalität gemessen. Ein Sample bezeichnet damit die gemessenen Daten je Lokalität. Im konkreten sind es SonarQube-Softwaremetriken und Annotationen zur Anzahl Fehler-indizierender nachfolgender Commits eines CommitPaths.
JupyterNotebook	Jupyter Notebook ist „ein Veröffentlichtungsformat für reproduzierbare computergestütze Arbeitsabläufe“ (Kluyver et al., 2016, übersetzt).

Softwaremetrik	„[Eine Software-Qualitäts-Metrik ist] eine Funktion, welche Software-Daten [Komponenten] als Eingabe und einen einzelnen numerischen Wert als Ausgabe besitzt, welcher als Grad interpretiert werden kann, in welchem [eine] Software[-Komponente] ein gegebenes Attribut, welches deren Qualität beeinflusst, besitzt.“ („IEEE Standard for a Software Quality Metrics Methodology“, 1994)
SonarQube	„SonarQube ist ein automatisches Codereview (Beurteilungs-) Werkzeug zur Detektierung von Fehlern, Schwachstellen und Code Smells in Quellcode. Es kann in einen existierenden Workflow integriert werden, um kontinuierliche Code-Inspektionen über Projekt-Branches und Pull-Requests zu ermöglichen.“[1]

[1] https://docs.sonarqube.org/9.1/, frei übersetzt (Stand: 15.11.2021)

5 Datenerhebung und Anwendung maschineller Lernverfahren

Dieses Kapitel behandelt das Vorgehen zur Datenerhebung und die Anwendung maschineller Lernverfahren zur Prädiktion Fehler-behebender nachfolgender Änderungen einer Datei. Die Daten wurden durch die im Kapitel 4 beschriebenen Komponenten erhoben. Die Umsetzung dieses Kapitel basiert im Wesentliche auf Module der Scikit-Learn-Bibliothek.

5.1 Lerndatensatz

Es wurden neun Lerndatensätze für jede Kombination aus Datensätzen der unabhängigen und abhängigen Variablen erzeugt. Jeder erhobene Datensatz enthält Daten für CommitPaths. Es wurden TypeScript-Dateien aus dem Verzeichnis *src* des Open Source Projektes *TypeScript* betrachtet. In den Kapiteln 5.1.1 und 5.1.2 wird der Datensatz definiert, auf dem die Experimente dieses Kapitel aufbauen. Der Datensatz besteht aus Features und Klassen für 11.080 Dateizustände aus 10.000 Commits.

5.1.1 Unabhängige Variablen

Zur Berechnung der drei Feature-Datensätze wurden je 48 Softwaremetriken der vergangen drei, fünf und zehn Dateizustände, inklusive des betrachteten Dateizustandes, einbezogen. Es wurden der Minimalwert, Maximalwert und arithmetische Durchschnitt aller Softwaremetriken aller betrachteten Dateizustände, die minimale und maximale Differenz, die arithmetische Durchschnittsdifferenz, die minimale relative, maximale relative und arithmetisch durchschnittliche relative Differenz (siehe Formel 5.1) aller Softwaremetriken zweier aufeinander folgender Dateizustände berechnet. Von den 432 möglichen Features je Datensatz wurden 199 entfernt, um null sowie +/- Infinity (Unendlich) -Werte im Datensatz zu vermeiden. Jeder Datensatz enthält somit 233 Features je Sample. In diesem

Kapitel wird der Datensatz verwendet, der die vergangenen fünf Dateizustände berücksichtigt.

$$\text{Relative Differenz} = \frac{x_2 - x_1}{x_1} \tag{5.1}$$

5.1.2 Abhängige Variable

Es wurden drei Label-Datensätze erzeugt. Je ein Datensatz wurde für jede Berücksichtigung der nächsten drei, fünf und zehn nachfolgenden Änderungen einer Datei erzeugt. Die zugeordnete Klasse ist die Summe der Fehler-Behebungs-indizierenden Dateiänderungen. Eine Dateiänderung wurde als Fehler-Behebungs-indizierend klassifiziert, wenn:

1. Die Commit-Message (Nachricht) Fehler-Behebungs-indizierende Schlagworte enthält. Es wurden die Schlagworte bug, fix, error und fail ohne Berücksichtigung der Groß- und Kleinschreibung einbezogen.
2. Der Commit nicht mehr als zwei Dateien, ohne Einbezug von Testdateien und ohne Beachtung gelöschter Dateien, enthält.
3. Der Commit kein Merge-Commit ist und
4. Die betrachtete Datei des Commits selbst keine Testdatei ist.

In diesem Kapitel wird der Datensatz verwendet, der die nächsten fünf Dateizustände berücksichtigt. Folgende Klassenverteilung enthielt der Datensatz:

Klasse	Samples
0	8671
1	2076
2	305
3	27
4	1
5	0

5.1.3 Datenexploration

Auf Basis der Diagramme einzelner Features und der dazugehörigen Klasse sind Tendenzen nicht sicher abzulesen (siehe Abb. 5.1). Es scheint, dass für das Feature *minValLines* eine höhere Klasse wahrscheinlicher bei einem kleineren minValLines-Wert ist. Das ist aus dem Graphen allerdings nicht sicher abzulesen, da nicht ersichtlich ist, ob schlichtweg die Häufigkeit der Samples in diesem Wertebereich höher ist und dadurch statistisch häufiger die Klasse 3 und 4 auftritt. Ähnliches gilt für die Interpretation der Plots für *maxValLines*, *meanDiffLines* und *meanRelDiffLines*. Im Allgemeinen lassen sich auf Basis solcher Plots, für den bestehenden Datensatz, durch Menschen, nur schlecht Klassifikationsregeln ableiten.

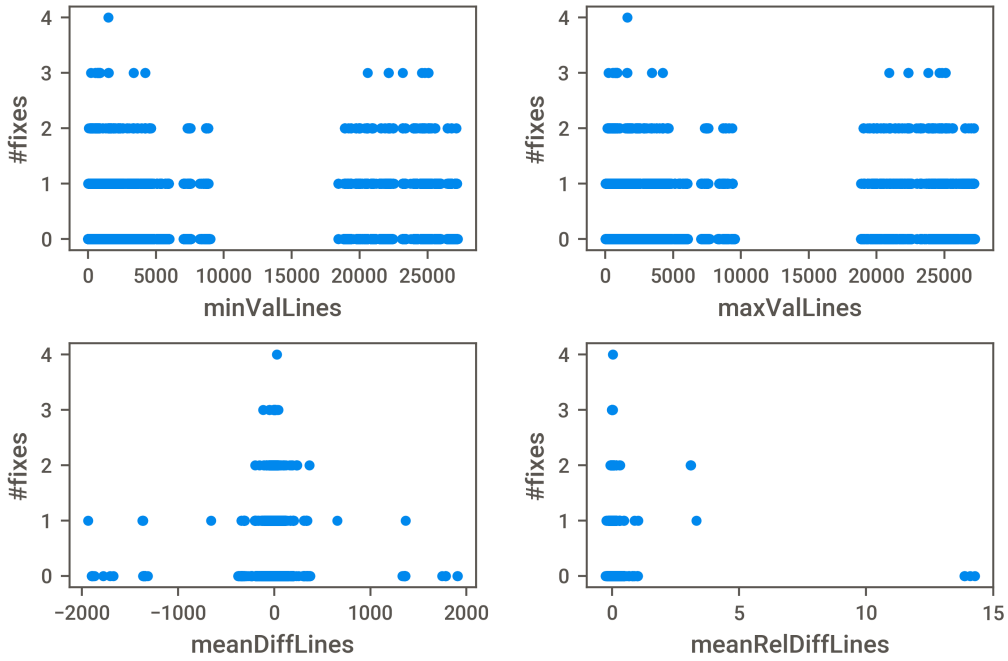

Abbildung 5.1: Klassenverteilung einzelner Features

5.2 Datentransformation

Die Features wurden in unverändeter und transformierter Form verwendet. Es wurde eine Z-Transformation der unveränderten Features durchgeführt. Mithilfe einer Principal Component Analysis, einem Feature-Reduction-Algorithmus, konnten 233 Z-transformierte Features auf 46 reduziert werden, sodass über 98% der Varianz der untransformierten Features mit den transformierten Features erklärt werden konnten. Es wurde ein binärer Datensatz aus den untransformierten Daten erzeugt, in dem alle Klassen größer eins auf eins abgebildet wurden. Es erfolgte

auch auf dem binären Datensatz eine Z-Transformation. Sowohl der untransformierte binäre als auch der untransformierte multi-class Datensatz wurden mit einer Min-Max-Skalierung auf den Wertebereich [0, 1] normalisiert. Die binären Datensätze werden in Kapitel 5.7 verwendet.

5.3 Modell- und Hyperparameterwahl

Es wurden mehrere Modelle verwendet. Für alle Modelle wurde ein Grid-Search über verschiedene Hyperparameter der Modelle auf den Trainingsdaten mit einer 10-fold cross-validation durchgeführt. Anschließend wurde das Modell unter Verwendung der besten Hyperparameter, gefunden durch Grid-Search, auf den Trainingsdaten trainiert.
Als Modelle wurden ein Decision-Tree-Klassifikator, ein Random-Forest-Klassifikator, ein Support-Vector-Machine-Klassifikator, ein Gradient-Boosting-Klassifikator und eine Logistische-Regression aus Scikit-Learn verwendet. Die Support-Vector-Machine verwendete die mittels Principal Component Analysis reduzierten Daten. Der Decision-Tree-Klassifikator, der Random-Forest-Klassifikator und der Gradient-Boosting-Klassifikator arbeiteten auf den untransformierten Daten. Die Logistic-Regression wurde sowohl auf den standardisierten wie auch auf den normalisierten Daten angewandt.

5.4 Evaluation

Es wurden 70% der Samples als Trainingsdatensatz und 30% als Testdatensatz verwendet. Da es nur sehr wenige Samples für einige Klassen gab, sind die macro average Bewertungen negativ verzerrt. Üblicherweise werden Samples, seltener Klassen, entfernt. Da diese Klassen aber Aufschluss über die Semantik der Modelle geben, wurden die Klasse drei und vier zunächst beibehalten. Zur Unterscheidung der Klassifikationsgüte zu Dummy-Klassifikatoren, ist die Betrachtung selten vorkommender Klassen relevant.

5.4.1 Baseline Klassifikatoren

Verschiedene einfache Klassifizierungsstrategien, sogenannte Dummy-Klassifikatoren, wurden zum Vergleich herangezogen. Verglichen wurden die Strategien constant für alle Klassen, stratified, most_frequent, prior und uniform (siehe Kapitel 3.1.13). Die besten Dummy-Klassifikator besaßen eine Accuracy von 78% und einen durchschnittlichen absoluten Fehler von etwa 0.25. Die pauschale Vorhersage der am häufigsten vorkommenden Klasse führte zu folgendem Ergebnis:

	precision	recall	f1-score	support
0	0.78	1	0.88	2588
1	0	0	0	642
2	0	0	0	85
3	0	0	0	8
4	0	0	0	1
accuracy			0.78	3324
macro avg	0.16	0.20	0.18	3324
weighted avg	0.61	0.2	0.18	3324
mean abs. error			0.2527	3324

Die stratified-Strategie erzielte die besten macro average Precision und Recall, dafür allerdings eine deutlich schlechtere Accuracy und einen relativ schlechten durchschnittlichen absoluten Fehler:

	precision	recall	f1-score	support
0	0.78	0.8	0.79	2588
1	0.22	0.2	0.21	642
2	0.04	0.04	0.04	85
3	0	0	0	8
4	0	0	0	1
accuracy			0.67	3324
macro avg	0.21	0.21	0.21	3324
weighted avg	0.65	0.67	0.66	3324
mean abs. error			0.3809	3324

5.4.2 Random-Forest-Klassifikator

Der Random-Forest-Klassifikator performte, ähnlich wie der Gradient-Boosting-Klassifikator, von allen manuell gewählten Modellen und getesteten Hyperparametern, am besten. Der Random-Forest-Klassifikator kann üblicher Weise schneller trainiert werden als der Gradient-Boosting-Klassifikator. Die anderen Klassifikatoren schnitten schlechter ab. Besonders interessant sind die relativ hohen Precision- und Recall-Werte für die Klassen zwei und drei. Beide Klassen besitzen einen geringen Support und sind damit schwer zu trainieren. Diese Klassen geben einen Hinweis darauf, dass nicht nur die Verteilung der Daten erlernt wurde, sondern dass tatsächlich in den Features Wissen über die Klassen vorhanden ist und erlernt wurde.

	precision	**recall**	**f1-score**	**support**
0	0.86	0.96	0.9	2588
1	0.63	0.38	0.48	642
2	0.29	0.12	0.17	85
3	0.4	0.25	0.31	8
4	0	0	0	1
accuracy			0.82	3324
macro avg	0.43	0.34	0.37	3324
weighted avg	0.8	0.82	0.8	3324
mean abs. error			0.1913	3324

Folgende Konfusionsmatrix wurde erzielt:

2477	110	1	0	0
375	246	21	0	0
37	35	10	3	0
3	0	3	2	0
1	0	0	0	0

Ein grafische Darstellung der Wichtigkeit einzelner Features (siehe Abb. 5.2) zeigt, dass viele Features in die Klassifikation mit einbezogen werden. Die zehn wichtigsten Features sind: meanValLines, maxValLines, meanValCognitiveComplexity, meanRelDiffLines, minValLines, meanDiffLines, maxRelDiffLines, minRelDiffLines, minDiffLines und maxValCognitiveComplexity, wobei mean für den arithmetischen Durchschnitt, min für das Minimum, max für das Maximum, diff für die Differenz und relDiff für die relative Differenz stehen. Es ist wahrscheinlich, dass einige dieser Features stochastisch voneinander abhängig sind. Derartige Untersuchungen werden zukünftigen Arbeiten überlassen. Die wichtigsten Features scheinen sowohl mit den Lines Of Code als auch mit der kognitiven Komplexität einer Datei verwandt zu sein. Dies assoziiert mit typischen Programmier-Richtlinien, welche die Vermeidung hoher kognitiver Komplexitäten und großer Dateien fordern. Das hier bestehende Modell berücksichtigt darüber hinaus die Veränderungen dieser Metriken über Dateiänderungen, welche in Commits manifestiert werden.

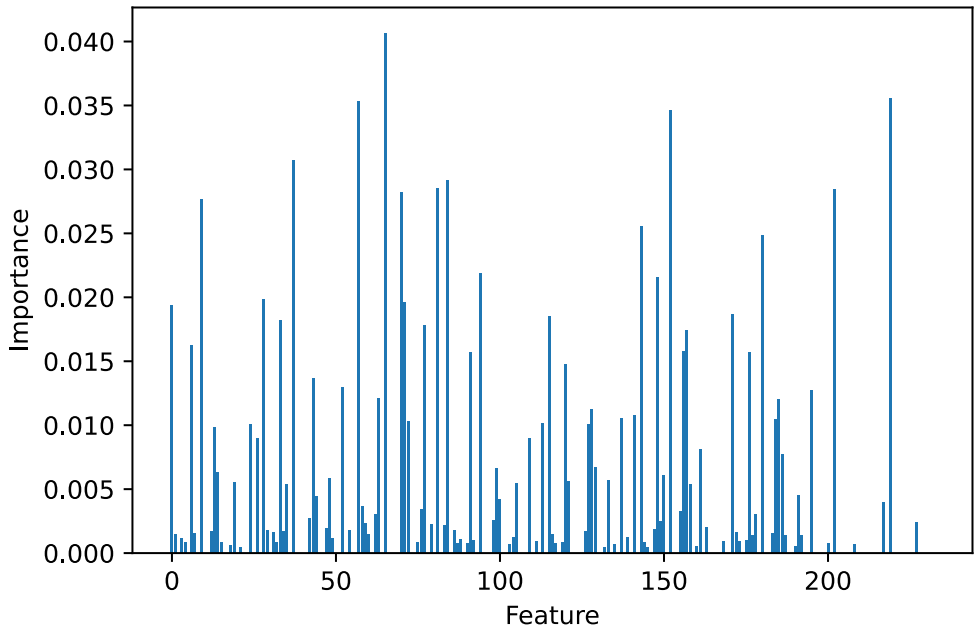

Abbildung 5.2: Wichtigkeit bestimmter Features

5.5 Automatisiertes maschinelles Lernen

Es wurden zwei Ansätze für automatisiertes maschinelles Lernen verwendet: TPOT und H2O. Aufgrund geringer Rechenkapazitäten konnten diese nicht ihr volles

Potenzial entfalten.

5.5.1 TPOT

TPOT wurde mehrere Stunden auf einem PC trainiert. Es ist denkbar, dass mit mehr Rechenaufwand bessere Ergebnisse mit TPOT erzielt werden.

	precision	**recall**	**f1-score**	**support**
0	0.87	0.95	0.91	2588
1	0.61	0.42	0.49	642
2	0.29	0.21	0.24	85
3	0.33	0.25	0.29	8
4	0	0	0	1
accuracy			0.82	3324
macro avg	0.42	0.36	0.39	3324
weighted avg	0.8	0.82	0.81	3324
mean abs. error			0.1898	3324

Folgende Konfusionsmatrix wurde erzielt:

2450	133	3	2	0
337	267	38	0	0
28	37	18	2	0
3	0	3	2	0
1	0	0	0	0

Die Klassifikationsgüte der von TPOT errechneten Pipeline liegt nah an der des Random-Forest-Klassifikators. Insbesondere waren die tatsächlichen Klassen für die vorhergesagten Klassen drei (und vier) exakt gleich häufig null, eins, zwei, drei und vier. Die Klasse vier ist kaum relevant, da es nur ein Sample im ganzen

Datensatz gab und damit kein Sample im Trainingsdatensatz vorkam.

Die Differenz der Tabellen zur Darstellung der Klassifikationsgüte des Random-Forest-Klassifikators und der TPOT-Pipeline zeigt gut, wie sich die Güte der beiden Vorgehensweisen unterscheiden. Ein positiver Wert steht für das bessere Abschneiden des Random-Forest-Klassifikators, ein negativer für das besser Abschneiden der TPOT-Pipeline, ausgenommen ist der durchschnittliche absolute Fehler, für welche die Regel invertiert gilt.

	precision	**recall**	**f1-score**	**support**
0	-0.02	0.01	-0.01	2588
1	0.02	-0.04	-0.02	642
2	-0.08	-0.14	-0.13	85
3	0.07	0	0.02	8
4	0	0	0	1
accuracy			0	3324
macro avg	0	-0.03	-0.03	3324
weighted avg	-0.01	0	-0.01	3324
mean abs. error			0.0027	3324

Insbesondere für die Klasse zwei scheint TPOT etwas besser zu sein, besonders die Verbesserung im Recall ist wünschenswert (siehe Kapitel 5.6, S. 116). Die Unterschiede der beiden Vorgehensweisen sind relativ gering. Die Aussagekraft bleibt aufgrund des geringen Supports für die Klasse drei gering.

5.5.2 H2O

Das automatische maschinelle Lernen mit H2O wurde acht Stunden auf einem Desktop-PC (CPU: i7 4790k, Kerne: 4, Threads: 8, Grundtaktfrequenz: 4GHz, Turbo-Taktfrequenz: 4.4GHz) trainiert. H2O ist darauf ausgelegt auf Clustern zu rechnen. Es ist zu erwarten, dass mit größeren Rechenkapazitäten und längeren Trainingszeit, bessere Ergebnisse erzielt werden können.

	precision	**recall**	**f1-score**	**support**
0	0.87	0.95	0.91	2588
1	0.64	0.43	0.51	642
2	0.41	0.19	0.26	85
3	0.43	0.38	0.4	8
4	0	0	0	1
accuracy			0.83	3324
macro avg	0.47	0.39	0.42	3324
weighted avg	0.81	0.83	0.81	3324
mean abs. error			0.1808	3324

Die Differenz der Tabellen zur Darstellung der Klassifikationsgüte des Random-Forest-Klassifikators und des durch H2O errechneten Modells zeigt, wie sich die Güte der beiden Vorgehensweisen unterscheiden. Die Tabelle ist wie in Kapitel 5.5.1 zu interpretieren. Während TPOT am besten für die Klasse drei abschneidet, dafür kaum Unterschiede in der Klasse drei zum Random-Forest-Klassifikator zeigt, schneidet das H2O-Modell sowohl für die Klasse zwei als auch für die Klasse drei besser als der Random-Forest-Klassifikator ab. Aufgrund des geringen Supports ist die Aussagekraft allerdings gering.

5.6 Interpretation der Klassifikationsgüte

Eine typische Abwägungsentscheidung und Interpretation der Klassifikationsgüte im Kontext des Einsatzbereichs ist die Priorisierung der Precision oder des Recalls. In diesem Fall dürfte es von Interesse sein einen höheren Recall der höheren Klassen zu erzielen, auch wenn dafür die Precision geringer ausfällt. Im Kontext der automatischen Fehleridentifikation ist es den Softwareentwicklern vermutlich wichtiger möglichst viele Fehler zu identifizieren unter allen der Klasse zugeordneten Dateien, geringer ausfällt. Da der Klassifikationsalgorithmus auch nahe liegende Klassen prädizieren kann, also z.B. die Klasse zwei, obwohl die Klasse drei korrekt wäre, ist der durchschnittliche absolute Fehler auch von Bedeutung. Die Aussagekraft dieser Metrik ist wiederum durch die Klasseninbalance eingeschränkt. Durch das häufige Auftreten der Klasse null fällt sie mehr ins

Gewicht, obwohl gerade die höheren Klassen vermutlich wichtiger wären. Die drei Vorgehensweisen zur Klassifikation, mittels TPOT, des H2O-Modells oder des Random-Forest-Klassifikator, sind alle geeignet zur Fehlerklassifikation anhand der hier erzeugten Daten. TPOT und H2O schneiden etwas besser ab und sollten daher bevorzugt werden.

5.7 Binäre Klassifikation

Da der Support der Klassen drei und vier äußerst gering und der Support der Klasse zwei nur moderat für den bestehenden Datensatz ausfällt, wurden experimentell alle Klassen über eins auf eins gesetzt. Es bleiben die Klassen null und eins übrig, also wird prädiziert, ob ein Fehler in den nächsten fünf Änderungen der Datei behoben wird oder nicht.

Für die binäre Klassifikation wurden ebenfalls einige Modelle gewählt und trainiert. Die Modelle erzeugen auf den Testdaten folgende ROC-Kurven: Das Modell *LR*

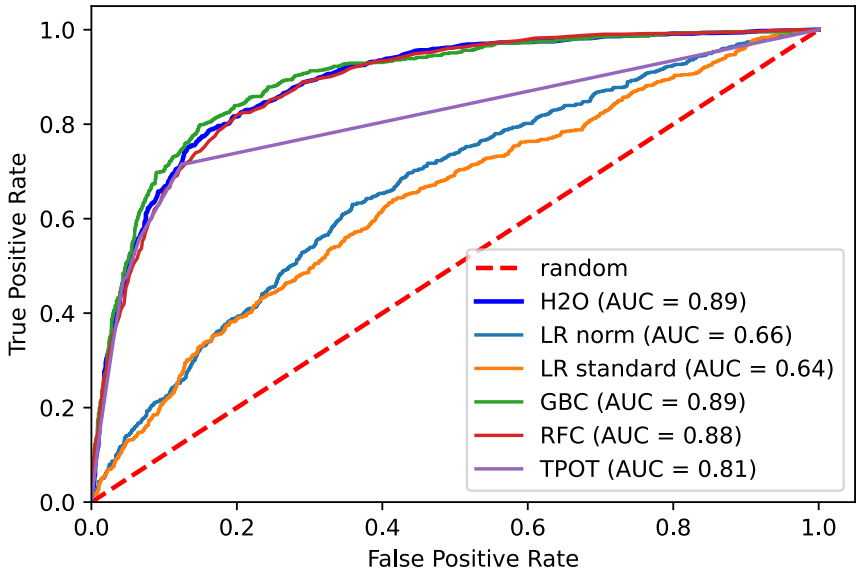

Abbildung 5.3: ROC-Kurven

norm steht für eine logistische Regression auf Min-Max-normalisierten Daten. Das Modell *LR standard* steht für eine logistische Regression auf standardisierten Daten. GBC steht für den Gradiend-Boosting-Klassifikator, RFC für den Random-Forest-Klassifikator. TPOT und H2O für die jeweiligen Modelle, welche durch TPOT und H2O erzielt wurden. TPOT wurde für die binäre Klassifikation kürzer trainiert, als für die n-äre Klassifikation, wodurch sich die vermutlich schlechteren Ergebnisse

erklären lassen. Auffällig ist, dass TPOT wenige Punkte im ROC-Graph besitzt. Für das mit TPOT errechnete Modell wurden keine Hyperparameter gefunden, sodass die Klassifikationsgrenzen so angepasst werden konnten, dass weitere Punkte im ROC-Graph entstehen. Diese Auffälligkeit steht für weitere Untersuchungen offen. Die rot gestrichelte Linie steht für ein Modell, das gleichmäßig und zufällig die Klassen null und eins vorhersagt. Alle Kurven, die über dieser Linie sind, performen besser als das Zufallsmodell. Dies trifft auf alle errechneten Modelle zu. Die logistische Regression auf normalisierten Daten erzielt eine bessere AUC als jene auf standardisierten Daten. Beide schneiden im Vergleich schlecht ab. Die errechneten Modelle Gradient-Boosting-Klassifikator, Random-Forest-Klassifikator und das mittels H2O errechnete Modell schneiden ähnlich und im Vergleich am besten ab.

6 Zusammenfassung und Ausblick

Zusammenfassung

Diese Arbeit befasst sich mit Korrelationsuntersuchungen zwischen quantifizierten Softwareartefakten und Fehlerauftreten mithilfe maschineller Lernverfahren. Es wurde ein nach Arc42 standardkonform dokumentiertes Open Source Framework publiziert, welches den Austausch von Entwicklungs- und Forschungsartefakten unter Forschenden motivieren soll. Die Weiterentwicklung zu einem Programm zur automatischen Fehleridentifikation wird ermöglicht. Für die Entwicklung der Architektur wurden konkrete Implementierungen vorgenommen, welche über die npm-Registry und Github publiziert wurden. Ziel ist die Erzeugung eines Lerndatensatzes mit möglichst wenigen Anforderungen an das zu untersuchende Projekt. Es wurden Dateien aus 10.000 Commits des öffentlich zugänglichen TypeScript-Projekts als Lokalitäten gemessen, mit SonarQube quantifiziert und anhand der Commit-Beschaffenheit annotiert. Eine Vielzahl an Projekten wird durch diese Implementierung unterstützt. SonarQube unterstützt bis zu 29 Programmiersprachen und Git ist ein weit verbreitetes Werkzeug in der Softwareentwicklung. Es wurden 233 Features über die letzten drei, fünf und zehn Änderungen einer Datei berücksichtigt, um die Anzahl der Fehlerbehebungen der nächsten drei, fünf und zehn Änderungen vorherzusagen. Alle Datensätze wurden veröffentlicht. Für die experimentelle Umsetzung des maschinellen Lernens wurde der Datensatz über die letzten fünf Änderungen zur Vorhersage der Fehlerbehebungen der nächsten fünf Änderungen verwendet. Verschiedene Verfahren des maschinellen Lernens wurden getestet, darunter automatisches maschinelles Lernen mit TPOT und H2O. Ein, mit H2O trainiertes, Modell konnte mit einer Accuracy von 83% die richtige Klasse vorhersagen, der durchschnittliche absolute Fehler betrug 0,18, der gewichtete Durchschnitt des F1-Scores betrug 81%, der ungewichtete, aufgrund des geringen Supports der Klassen drei und vier, 42%. Es wurden alle Klassen über eins auf eins gesetzt und binäre Klassifikatoren entwickelt. Die besten Modelle, ein durch H2O ermitteles Modell und ein Gradient-Boosting-Klassifikator, erzielten eine AUC der ROC-Kurve von 0,89. Das Grid-Search des Random-Forest-Klassifikators erfolgte zügig. Der RFC könnte daher ein geeignetes Modell für das erste Training ähnlicher Datensätze sein.

Ausblick

Eine Reihe weiterer Untersuchungen und Implementierungen werden zukünftigen Arbeiten überlassen:

1. Unterstützung weiterer Lokalitätsklassen, wie z.B. Architektur-Artefakte.
2. Unterstützung weiterer Metriken, wie die Änderungsfrequenz, die Uhrzeit der Änderung oder die absolute Zeitdifferenz zur letzten Änderung einer Datei.
3. Quantifizierung von Zeitreihendaten und Verwendung passender Verfahren des maschinellen Lernens. Hintergrund: Diese Arbeit berechnete im Quantifizierungsprozess Aggregationen von Zeitreihendaten, konkret von Softwaremetriken mehrerer Dateizustände. Es gibt maschinelle Verfahren, die anhand der Zeitreihendaten eine Vielzahl verschiedener Untersuchungen durchführen können.[1]
4. Unterstützung von Cluster-Computing des Quantifizierungsprozesses.
5. Quantifizierung verschiedener Softwareprojekte.
6. Manuelle Annotation von Lokalitäten.
7. Multiklassen-Annotation von Lokalitäten, z.B. nach Schweregrad oder Art des Fehlers.
8. Annotation von Lokalitäten anhand von Bugtracker-Systemen.
9. Untersuchungen zur Identifikation des Zeitpunkts des Auftretens von Fehlern.
10. Betrachtung von Netzwerkanalysen über die Dateien und Änderungen durch Autoren.
11. Multivariate Untersuchungen der Features.
12. Anwendung weiterer Verfahren des maschinellen Lernens, wie moderne Feature-Engineering-Verfahren (Horn et al., 2019).
13. Untersuchungen zu den weiteren veröffentlichten Datensätzen. Hintergrund: Diese Arbeit untersuchte nur einen der neun veröffentlichten Datensätze.
14. Erzeugung und Evaluierung von Ensemble-Modellen verschiedener Quantifizierungs- oder Fehleridentifikations-Strategien.
15. Erzeugung und Evaluierung von Ensemble-Modellen über verschiedene, hierarchische Lokalitätsklassen. Beispiel: Ensemble eines Modells zur Vorhersage von Fehlern in Dateien und eines zur Vorhersage von Fehlern einzelner Funktionen einer Datei.

[1] https://tsfresh.readthedocs.io/en/latest/text/introduction.html (Stand: 29.11.2021)

16. Längeres Training mittels TPOT.
17. Längeres Training auf Clustern mittels H2O.
18. Untersuchungen der Generalisierbarkeit der errechneten Modelle auf verschiedene Softwareprojekte.
19. Vollautomatische Unterstützung der Lernphase (siehe Kapitel 4.7, S. 61) eines Projekts, mit dem Ergebnis eines Fehleridentifizierungsprogramms, welches auf das Projekt angewandt werden kann. Hintergrund: Es ist ungeklärt, ob auf spezifische Projekte antrainierte Modelle auf andere Projekte generalisierbar sind. Wird ein Modell für das bestehende Projekt erzeugt, so trainiert es voraussichtlich Projekt-spezifische Fehler.
20. Untersuchungen zu Verfahren des bestärkenden Lernens (reinforcement learning). Während des Entwicklungsprozesses könnten Anwender des Fehleridentifizierungsprogramms angeben, wie gut oder schlecht eine Modell-Prädiktion war. Ein Modell könnte mittels bestärkendem Lernen solche Eingaben zur Selbstoptimierung verwenden.
21. Verbesserung qualitativer Merkmale, wie der Nutzbarkeit (Usability), der Zuverlässigkeit oder des Zeitverhaltens.

Anhang

A Einführung: Dependency Injection mit InversifyJS

InversifyJS

„InversifyJS ist ein leichtgewichtiger Inversion of Control (IoC) Behälter für TypeScript- und JavaScript-Applikationen. Ein IoC-Behälter nutzt Klassenkonstruktoren zur Identifizierung und Injektion derer Abhängigkeiten. “ (Jansen, 2021, frei übersetzt)

Installation

Durch

```
npm i -D inversify reflect-metadata
```

werden die benötigten Abhängigkeiten installiert. Das Folgende bezieht sich auf die Version `5.0.1`. Es wird TypeScript in einer Version >= 2.0, sowie die Aktivierung der Kompilieroptionen `experimentalDecorators`, `emitDecoratorMetadata`, `types` und `lib` benötigt. Diese können wie folgt in der tsconfig.json angegeben werden:

```
{
    "compilerOptions": {
        "target": "es5",
        "lib": ["es6"],
        "types": ["reflect-metadata"],
        "module": "commonjs",
        "moduleResolution": "node",
        "experimentalDecorators": true,
        "emitDecoratorMetadata": true
    }
}
```

Der polyfill `reflect-metadata` muss genau einmal in der ganzen Applikation importiert werden. Beispiel: main.ts

```
import "reflect-metadata";
```

Beispiel Im Folgenden wird das Basics-Beispiel der InversifyJS-Dokumentation aufgezeigt unter Korrektur eines Dokumentationsfehlers. Die offizielle InversifyJS-Dokumentation besitzt einen Fehler: Statt der Funktion Symbol, wird Symbol.for genutzt. Mehrfachausführung der Funktion Symbol.for mit gleichem Parameter führt allerdings zur Generierung gleicher Schlüssel. Jansen, 2021

1. Schnittstellen und Typen

```
// file interfaces.ts

```

```
3  export interface Warrior {
4      fight(): string;
5      sneak(): string;
6  }
7
8  export interface Weapon {
9      hit(): string;
10 }
11
12 export interface ThrowableWeapon {
13     throw(): string;
14 }
```

Typischerweise wird auf Abstraktionen, hier Interfaces, gearbeitet. Diese Schnittstellen werden von Klassen verwendet. Die konkrete Implementierung dieser Schnittstellen ist für die verwendenden Klassen zur Implementierungszeit nicht relevant.

```
1  // file types.ts
2
3  const TYPES = {
4      Warrior: Symbol("Warrior"),
5      Weapon: Symbol("Weapon"),
6      ThrowableWeapon: Symbol("ThrowableWeapon")
7  };
8
9  export { TYPES };
```

Die Variable `TYPES` ist ein Objekt zur Namensauflösung. Solche Objekte werden für jedes Plugin erzeugt, sodass diese über die Schlüssel der Types konfiguriert werden können. Das bugFinder-framework-Paket gibt ebenfalls Types vor, welche injiziert werden, sodass die einzelnen Prozesse der Pipeline ausgeführt werden können (siehe Kapitel 4.6, S. 60). Das Attribut `Warrior` des Objekts wird für zu injizierende Elemente, wie Attribute einer Klasse, angegeben, um eindeutig zu kennzeichnen, dass genau das Objekt, das für `Warrior` angegeben wird, zu injizieren ist. Die Funktion Symbol(arg: string) generiert einen einzigartigen (unique) Wert. Der wiederholte Aufruf der Funktion Symbol("Warrior") führt demnach zur Generierung zweier verschiedener Schlüssel. InversifyJS nutzt diese Schlüssel als Identifizierer zur Laufzeit zur Auflösung der zu injizierenden Instanzen. Statt Symbol(arg: string) können auch Klassen oder Zeichenketten angegeben werden.

2. Abhängigkeits-Deklaration mittels @injectable und @inject

```
1  // file entities.ts
2
3  import { injectable, inject } from "inversify";
4  import "reflect-metadata";
5  import { Weapon, ThrowableWeapon, Warrior } from "./interfaces"
6  import { TYPES } from "./types";
```

```
7
8  @injectable()
9  class Katana implements Weapon {
10     public hit() {
11         return "cut!";
12     }
13 }
14
15 @injectable()
16 class Shuriken implements ThrowableWeapon {
17     public throw() {
18         return "hit!";
19     }
20 }
21
22 @injectable()
23 class Ninja implements Warrior {
24
25     private _katana: Weapon;
26     private _shuriken: ThrowableWeapon;
27
28     public constructor(
29         @inject(TYPES.Weapon) katana: Weapon,
30         @inject(TYPES.ThrowableWeapon) shuriken: ThrowableWeapon
31     ) {
32         this._katana = katana;
33         this._shuriken = shuriken;
34     }
35
36     public fight() { return this._katana.hit(); }
37     public sneak() { return this._shuriken.throw(); }
38
39 }
40
41 export { Ninja, Katana, Shuriken };
```

Durch die Decorators `@injectable()` und `@inject()` werden Klassen, welche injizierbar sind und Werte, welche injizierten werden sollen, angegeben. Bei den durch `@injectable` gekennzeichneten Klassen handelt es sich um Implementierungen der vorher angegebenen Abstraktionen (Interfaces). Der `@inject`-Decorator erhält als Parameter den Identifizierer für das jeweilige Interface der zu injizierenden Variable. Alternativ zur Konstruktoren-Injektion kann auch Attribut-Injektion verwendet werden:

```
1 @injectable()
2 class Ninja implements Warrior {
3     @inject(TYPES.Weapon) private _katana: Weapon;
4     @inject(TYPES.ThrowableWeapon) private _shuriken:
          ThrowableWeapon;
5     public fight() { return this._katana.hit(); }
6     public sneak() { return this._shuriken.throw(); }
```

```
}
```

3. Erzeugung und Konfiguration eines Containers

```
// file inversify.config.ts

import { Container } from "inversify";
import { TYPES } from "./types";
import { Warrior, Weapon, ThrowableWeapon } from "./interfaces";
import { Ninja, Katana, Shuriken } from "./entities";

const myContainer = new Container();
myContainer.bind<Warrior>(TYPES.Warrior).to(Ninja);
myContainer.bind<Weapon>(TYPES.Weapon).to(Katana);
myContainer.bind<ThrowableWeapon>(TYPES.ThrowableWeapon).to(
    Shuriken);

export { myContainer };
```

Ein Container (Behälter) enthält die Konfiguration eines Programms. Es wird angegeben, welche Typen durch welche Instanzen realisiert werden. Die Container-API bietet hierbei verschiedene Möglichkeiten solche `bindings` anzugeben. Im Beispiel wird durch die Funktion `bind[...].to(Klasse)` das Verbinden eines Typs mit einer Klasse und der damit impliziten Instanziierung dieser Klasse durch die Container-API angegeben. Die hier angegebene Klasse kann selbst Abhängigkeiten besitzen, welche alle im Container gesetzt werden müssen, damit InversifyJS sie instanziieren kann. Es wird empfohlen diese `bindings`, also die Konfiguration, in einer einzelnen Datei anzugeben. Jansen, 2021

4. Auflösung der Abhängigkeiten

```
import { myContainer } from "./inversify.config";
import { TYPES } from "./types";
import { Warrior } from "./interfaces";

const ninja = myContainer.get<Warrior>(TYPES.Warrior);

expect(ninja.fight()).eql("cut!"); // true
expect(ninja.sneak()).eql("hit!"); // true
```

Durch Aufruf der `get<T>`-Funktion der Container-API wird eine Abhängigkeit aufgelöst. Laut offizieller Dokumentation, sollte diese Auflösung lediglich in der Kompositionswurzel des Programms erfolgen, „um das *service locator anti-pattern* zu vermeiden“. (Jansen, 2021)

Literaturverzeichnis

Norm ISO/IEC 25010. (2011). *ISO/IEC*.

Arisholm, E. & Briand, L. C. (2006). Predicting Fault-Prone Components in a Java Legacy System. *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, 8–17. https://doi.org/10.1145/1159733.1159738

Balzert, H. (1998). *Lehrbuch der Softwaretechnik, Teil 2: Softwaremanagement, Software-Qualitaetssicherung, Unternehmensmodellierung*. Spektrum Akademischer Verlag.

Bass, L., Clements, P. & Kazman, R. (2003). *Software Architecture in Practice (2nd Edition)*. Addison-Wesley Professional.

Clements, P., Kazman, R. & Klein, M. (2002). *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley. https://books.google.de/books?id=DV917BZ9RAgC

Erman, L., Hayes-Roth, F., Lesser, V. & Reddy, R. (1980). The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *ACM Comput. Surv.*, *12*, 213–253. https://doi.org/10.1145/356810.356816

Fawcett, T. (2006). Introduction to ROC analysis. *Pattern Recognition Letters*, *27*, 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

Fowler, M. (2004). Inversion of Control Containers and the Dependency Injection pattern. https://martinfowler.com/articles/injection.html

Horn, F., Pack, R. & Rieger, M. (2019). The autofeat Python Library for Automated Feature Engineering and Selection. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 111–120.

Hruschka, D. P. & Starke, D. G. ((Stand: 23.11.2021)). arc42 template. https://arc42.org/

IEEE SMC Maschinelles Lernen. (2020). http://www.ieeesmc.org/technical-activities/cybernetics/machine-learning

IEEE Standard for a Software Quality Metrics Methodology. (1994). *IEEE Std 1061-1998*, 1–32. https://doi.org/10.1109/IEEESTD.1994.9055496

Jansen, R. H. (2021). *InversifyJS - Dependency Injection*. https://github.com/inversify/InversifyJS

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S. & Willing, C. Jupyter Notebooks – a publishing format for reproducible computational workflows (F. Loizides & B. Schmidt, Hrsg.). In: In *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (F. Loizides & B. Schmidt, Hrsg.). Hrsg. von Loizides, F. & Schmidt, B. IOS Press. 2016, 87–90. https://doi.org/10.3233/978-1-61499-649-1-87.

Lalanda, P. (1997). Two complementary patterns to build multi-expert systems.

Le, T. T., Fu, W. & Moore, J. H. (2020). Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, *36*(1), 250–256.

LeDell, E. & Poirier, S. (2020). H2O AutoML: Scalable Automatic Machine Learning. *7th ICML Workshop on Automated Machine Learning (AutoML)*. https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf

Liu, K., Kim, D., Bissyande, T. F., Yoo, S. & Le Traon, Y. (2018). Mining Fix Patterns for FindBugs Violations. *IEEE Transactions on Software Engineering*, 1–1. https://doi.org/10.1109/TSE.2018.2884955

Malich, S. (2008). *Qualität von Softwaresystemen - Ein pattern-basiertes Wissensmodell zur Unterstützung des Entwurfs und der Bewertung von Softwarearchitekturen*. Springer-Verlag.

Mavin, A., Wilkinson, P., Harwood, A. & Novak, M. (2009). Easy Approach to Requirements Syntax (EARS). *2009 17th IEEE International Requirements Engineering Conference*, 317–322. https://doi.org/10.1109/RE.2009.9

McCabe, T. (1976). A Complexity Measure. *IEEE Transactions on Software Engineering*, *SE-2*(4), 308–320. https://doi.org/10.1109/TSE.1976.233837

Mockus & Votta. (2000). Identifying reasons for software changes using historic databases. *Proceedings 2000 International Conference on Software Maintenance*, 120–130. https://doi.org/10.1109/ICSM.2000.883028

MongoDB Dokumentation. (eingesehen am 15.11.2021). https://docs.mongodb.com/manual/introduction/

Ostrand, T. J., Weyuker, E. J. & Bell, R. M. (2005). Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, *31*(4), 340–355.

Ostrand, T. J. & Weyuker, E. J. (2010). Software Fault Prediction Tool. *Proceedings of the 19th International Symposium on Software Testing and Analysis*, 275–278. https://doi.org/10.1145/1831708.1831743

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Robert C., M. (2003). Agile Software Development, Principles, Patterns and Practices, 127–131.

Shin, Y., Maneely, A., Williams, L. & Osborne, J. A. (2010). Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *IEEE*. https://doi.org/10.1109/TSE.2010.81

SonarQube. (Stand 23.11.2021). https://www.sonarqube.org/

Weyuker, E. J., Ostrand, T. J. & Bell, R. M. (2008). Comparing Negative Binomial and Recursive Partitioning Models for Fault Prediction. *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, 3–10. https://doi.org/10.1145/1370788.1370792