Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Software Engineering

Entwicklung eines Tools zur Suche nach relevanten Diskussionen aus Github Issue-Trackern für die qualitative Forschung

Robert Selack

Matrikelnummer: 4774775

robfu@zedat.fu-berlin.de

Betreuer: Victor Brekenfeld

Eingereicht bei: Prof. Dr. Lutz Prechelt

Zweitgutachter: Prof. Dr. Claudia Müller-Birn

Berlin, 21. Oktober 2020

Vorwort

In der vorliegenden Bachelorarbeit wird die gewohnte männliche Sprachform bei personenbezogenen Wörtern verwendet, um eine leichtere Lesbarkeit zu erreichen. Demnach sind für die erleichterte Sprachverwendung jegliche Benennungen als genderneutral zu verstehen.

Abstract

Background. In software development, decisions are often made that demand a high price. These include so-called technical debts. To understand and avoid these debts, qualitative research is conducted. This field of research is still in its infancy.
Goal. The goal of this thesis is therefore to advance this research area. In the context of this work, a tool will be developed to provide data sets relevant for qualitative research purposes in the form of discussions about the handling of technical debt.
Methods. To achieve this, the tool uses the "Google Search JSON API", "Github APIänd a SSQLite database". The projects of Github are used here as a rough data set. Significant keywords are fed into the Google search engine and filter the projects for relevant discussions. These results are stored in a local SQLite-database and support the user with follow-up work.
Results. Through manual testing and quality control, it has been concluded that the use of keywords leads to high-quality hits. In addition, a tool has been developed based on these results that can be used for qualitative research.
Conclusions. The tool created in this thesis should be able to be used effectively in this scope. By using different metrics, the tool can further gain effectiveness and serve as a foundation for further tools.

Zusammenfassung

Hintergrund. Bei der Softwareentwicklung werden häufig Entscheidungen getroffen, die einen hohen Preis fordern. Dazu zählen sogenannte technischen Schulden („Technical Debt“). Um diese Schulden verstehen und vermeiden zu können, wird qualitative Forschung betrieben. Dieses Forschungsgebiet befindet sich noch am Anfang.
Ziel. Das Ziel dieser Arbeit ist es daher diesen Forschungsbereich voranzubringen. Im Rahmen dieser Arbeit wird dabei ein Tool entwickelt, das für qualitative Forschungszwecke relevante Datensätze in Form von Diskussionen über den Umgang mit technischen Schulden zu liefern.
Methoden. Um dies zu erreichen, verwendet das Tool die „Google Search JSON API“, „Github API“ und eine „SQLite-Datenbank“. Die Projekte von Github werden hier als grober Datensatz genutzt. Signifikante Schlüsselwörter werden der Google-Suchmaschine eingespeist und filtern die Projekte nach relevanten Diskussionen. Diese Treffer werden in eine lokale SQLite-Datenbank gespeichert und unterstützen mit Nachbereitung den Anwender bei der Untersuchung.
Ergebnisse. Durch manuelle Tests und Qualitätskontrollen konnte festgestellt werden, dass die Verwendung von Schlüsselwörtern zu hochwertigen Treffern führt. Zudem ist darauf basierend ein Tool entwickelt worden, das für die qualitative Forschung eingesetzt werden kann.
Schlussfolgerungen. Das in dieser Arbeit erstellte Tool soll in der Lage sein, effektiv im Anwendungsbereich genutzt zu werden. Durch die Verwendung von unterschiedlichen Metriken kann das Tool weiterhin an Effektivität gewinnen und als Fundament für weitere Tools dienen.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 21. Oktober 2020

Robert Selack

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Der Begriff „Technical Debt“ ist eine in 1992 von Ward Cunningham eingeführte Metapher [1] und findet seine Anwendung im Bereich des Software Engineerings. Bei der Softwareentwicklung kommt es oft zu bewussten oder unbewussten Entwurfsentscheidungen. Dabei handelt es grundsätzlich von zwei unterschiedlichen Ansätzen. Eine ist die elegantere und zeitaufwendige Lösung, die langfristig gesehen wenig Modifikation benötigt. Die andere ist eine schnell umsetzbare und zweckmäßig ausreichende Lösung, die in der Zukunft jedoch vielen Änderungen unterliegen wird. Beim Vorziehen der schnellen gegenüber der aufwendigeren Lösung kommt es zur Entstehung von technischen Schulden. Zur Tilgung dieser Schulden muss zusätzliche Zeit aufgewendet werden, um die resultierenden Konsequenzen zu beheben. Durch erzeugte Abhängigkeiten im Projekt kann der benötigte Zeitaufwand weiter ansteigen, je länger die Schulden im Entwicklungsprozess verweilen. Um diese Schulden zu minimieren, ist Forschung nicht nur auf Quelltextebene der Projekte notwendig. Besonders codeexterne Verhaltensmuster der Projektangehörigen müssen untersucht werden. Dabei soll unter anderem herausgefunden werden, wie Entwickler über technische Schulden denken oder wer besagte Entwurfsentscheidungen nach welchen Abwägungen trifft.

1.1 Motivation

Technische Schulden im Quelltext ausfindig zu machen, ist mittlerweile dank vielerlei Tools wie Ptidej [2], CLIO [3] oder FindBugs [4] unproblematisch. Dabei kann gezeigt werden, ob der verwendete Quelltext Schulden besitzt und wie groß der Zeitaufwand zur Tilgung dieser Verschuldungen wäre [5]. Das qualitative Forschungsgebiet benötigt jedoch keinen verschuldeten Quelltext, sondern Diskussionen zur Bewältigung, Entstehung und Vermeidung der Schulden. Dabei ist diese Art von Informationsbeschaffung innerhalb eines Projekts wesentlich komplizierter, da noch kein entsprechendes Tool existiert. Mit dieser Arbeit wird kein Tool entwickelt, das mit hundertprozentiger Genauigkeit dem Anwender jene Daten liefert. Stattdessen ist eine Anwendung umgesetzt worden, die dem Benutzer voraussichtlich relevante Diskussionen aus Projekten ausgibt. Dabei wird von Methoden profitiert, die zur Annähe-

rung an die gewünschten Datensätze führt. Zusätzlich werden die Diskussionen zur schnelleren und vereinfachten Analyse aufbereitet ausgegeben.

1.2 Anmerkung

Im Verlauf dieser Arbeit kam es zu einem Wechsel des umgesetzten Lösungsansatzes. Anfangs ist es meinerseits aufgrund eines nicht eindeutigen Verständnisses über die gewünschten Ergebnisse zu einem anders orientierten Gedankengang gekommen. Dieser entwickelte sich im Hinblick auf das Endprodukt in die richtige Richtung, war jedoch letztlich zu ungenau und breit gefächert realisiert worden. Im Bearbeitungszeitraum von insgesamt zwölf Wochen zog sich diese Phase der anfänglichen Umsetzung bis in die siebte Woche. Am Ende dieser Phase ist eine Zwischenpräsentation dieser Arbeit gehalten worden, in der der Stand zur bisher geleisteten Umsetzung gezeigt und konstruktive Kritik gesammelt wurde. Dabei zeigte sich, dass sich die Umsetzung in eine inkorrekte Richtung entwickelt hatte. Nach dieser Erkenntnis wurde während der Präsentation die Hauptproblematik erneut zusammen erörtert. Außerdem sind einige neue mögliche Ansätze und Ideen diskutiert worden, um mithilfe dieser nun in die richtige Richtung zu gelangen. Ab dieser Präsentation kam es zu dem genannten Wechsel im Lösungsansatz. Die Inhalte dieser geschriebenen Arbeit handeln dabei stets vom neuen Lösungsansatz, sofern nicht anders vermerkt. Einige Sektionen dieser Arbeit befassen sich, aufgrund von Dokumentationsgründen, ebenfalls mit der anfänglichen Phase. Diese werden im entsprechenden Titel kenntlich gemacht. Der anfängliche Lösungsansatz wird mit „Open Source Code Tool“(OSCTool) gekennzeichnet und der aktuelle Ansatz wird mit „Query Issue Finder Tool“vermerkt (QIFTool).

1.3 Aufbau der Arbeit

In diesem Abschnitt wird ein kurzer Ausblick auf den restlichen Teil der Arbeit gegeben. Sektion zwei 2 befasst sich mit dem Stand und dem Entwicklungsprozess des qualitativen Forschungsbereich und nennt im Anschluss einige verwandte Arbeiten. Im Abschnitt drei 3 werden einige Lösungsansätze beschrieben und daraufhin untersucht. Zudem wird der Entscheidungsprozess dargestellt, indem die Ansätze miteinander abgewogen werden. Der vierte Abschnitt 4 handelt von der Umsetzung des

entwickelten Tools. Es wird auf die verwendeten Bibliotheken, dem Reifegrad und der Qualitätssicherung eingegangen. In der Problembewältigung aus Sektion fünf 5 werden interessante Probleme und deren Bewältigung erläutert. Im Rahmen der gesamten Arbeit handelt das sechste Kapitel 6 von der Durchführung und beschreibt das Vorgehen einzelner Zeitabschnitte. Das siebte und letzte Kapitel 7 befasst sich mit dem resultierenden Fazit und Ausblick dieser Arbeit.

2 Stand der Forschung und verwandte Arbeiten

In dieser Sektion wird auf den aktuellen Stand im qualitativen Forschungsbereich und den wissenschaftlichen Arbeiten eingegangen, die sich inhaltlich der eigenen Arbeit nähern.

2.1 Stand der Forschung

Für die Verwaltung von Softwareprojekten wurden Versionskontrollsysteme entwickelt. Sie verwenden sogenannte Repositories für die Archivierung von gleichzeitig unterschiedlichen Versionen eines Projekts. Repositories sind eine bereits seit 1982 existierende Datenstruktur, die es Entwicklern erlaubt, Projekte samt ihrer Ordnerstruktur und Dateien abzuspeichern [6]. Seit den jährlichen Konferenzen von „Mining Software Repositories“ [7] wird kontinuierlich angestrebt, diese Datenstruktur nicht ausschließlich für reine Archivierungszwecke zu nutzen. Sie stellen einen großen Mehrwert für Bildungszwecke dar. Unter anderem sollen durch verfolgbare Informationen auf die Entscheidungen innerhalb eines Projekts und dessen Verlauf geschlossen werden. Diese Schlussfolgerungen sollen zu einem empirischen Verständnis führen, welches zu einer verbesserten Planung von und einem verbesserten Umgang mit zukünftigen Projekten beitragen soll [8]. Während der Recherche sind keine wissenschaftlichen Arbeiten aufgetreten, die auf den Fortschritt der qualitativen Forschung schließen. Dabei wurden jedoch Ausarbeitungen und Tools für die quantitative Forschung gefunden. Daraus ist zu schließen, dass sich das qualitative Forschungsgebiet für „Technical Debt“ am Anfang befindet.

2.2 Verwandte Arbeiten

In diesem Abschnitt werden einige Arbeiten, die dem Ansatz dieser Arbeit ähneln, genannt und kurz erläutert.
Das „*Github-Torrent-Projekt*“ (GHTorrent) ist eine Abbildung von Github und dessen Inhalten in Form einer gigantischen Datenbank [9]. Sie besitzt Einträge seit 2012 und setzt sich zum Ziel letztlich eine vollständige Abbildung zu erstellen. Außerdem ermöglicht sie den Zugriff auf jegliche Metainformationen und wird sowohl für interne als auch für externe Anwender genutzt, um Datenanalysen zu betreiben.
„*PyDriller*“ ist ein Framework für Python zum Extrahieren sinnvoller Informationen von Softwareprojekten [10]. Es kann auf jeglichen verteilten Versionskontrollsystemen (GIT) zugreifen. Zudem ermöglicht es das einfache exportieren von Metainformationen in das CSV-Format.
Das Tool „*CVSAnalY* “ [11] beschreibt eine ähnliche Vorgehensweise, indem es sich auf codebezogene Informationen von GIT bezieht und diese im Nachhinein in eine Datenbank abspeichert. Es unterstützt einige Programmiersprachen, mit eigen erstellten sowie berechneten Metriken, die über installierte Erweiterungen aus dem Quelltext extrahiert werden können.
Das Projekt „*Software Heritage*“ [12] stellt es sich zur Aufgabe jede öffentlich zugängliche Software zu sammeln, zu archivieren und miteinander zu teilen. Dabei wird nicht nur die aktuellste Version einer Software archiviert, sondern auch die vollständige Historie der jeweiligen Projekte. Mit dem gesammelten Datensatz wird einem ebenfalls die Möglichkeit geboten dessen Projekte nach Metainformationen zu filtern.
In der wissenschaftlichen Ausarbeitung *„Mining threat intelligence about open-source projects and libraries from code repository issues and bug reports“* [13] wurden Repositories von Open Source-Projekten verwendet, um unter anderem über dessen Issues und Programmfehlermeldungen auf mögliche Störungen und Gefährdungen für das Projekt zu schließen. Diese Schlussfolgerungen sollen dabei helfen informiertere Entscheidungen für den Verlauf des Projektes zu fällen.
Die wissenschaftliche Ausführung von *Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt 2017* befasst sich mit der Erkennung von selbst zugegebenen technischen Schulden (self admitted techincal debt) auf Grundlage von Kommentaren der Entwickler innerhalb des Quelltexts [14]. Der Fokus liegt hierbei,

im Gegensatz zu anderen genannten verwandten Arbeiten, also bei der Analyse von Sprache als Kommunikationsmedium in Kommentarform, anstelle von syntaktischem Code.
Diese Arbeit hingegen befasst sich, ähnlich wie die letztgenannte Ausarbeitung, mit der Kommunikation als Indikator. In diesem Fall jedoch für das allgemeine Auftreten von technischen Schulden. Zudem wird hierbei nicht auf den Quelltext des Projektes eingegangen, sondern auf die Issues, die die Kommunikationsschnittstelle der Entwickler bilden. Das Extrahieren von nach Metainformationen aus Repositories findet auch seinen Nutzen, um unterstützend zur Analyse auf hochwertigere Treffer zu kommen.

3 Lösungsansatz

Im folgenden Abschnitt werden unterschiedliche Lösungsansätze vorgestellt und anschließend abgewogen, um die Wahl des aktuellen Lösungsansatzes zu begründen. Bei den Ansätzen

3.1 QIFTool

Für die gewünschten Ergebnisse in Form von Diskussionen wird der Datensatz von „Github“ verwendet. Dieser netzbasierte Dienst, welcher zur verteilten Versionsverwaltung für Softwareentwicklungsprojekte genutzt wird [15], bietet eine enorme Menge an zugänglichen „Open Source“-Projekten. Die Lizenz dieser Projekte ermöglicht eine freie Verwendung, Änderung und Verteilung derer Projektarchive. Das entscheidende dabei ist jedoch, dass die Kommunikation zwischen den involvierten Entwicklern auf Github ebenfalls öffentlich zugänglich gemacht ist. Dazu werden sogenannte „Issues“ verwendet und beschreiben einen zu tätigen Aufwand, dem ein konkretes Thema zugeordnet wird. Sie werden von Entwicklern erstellt und tragen nach der Bewältigung zur Verbesserung des Projekts bei. Dabei bilden sie die Kommunikationsschnittstelle der Entwickler, um über eine festgelegte Thematik zu diskutieren. Github besitzt einen riesigen Datensatz von mehr als 50 Millionen Nutzern, 100 Millionen Projekten [16] und 550 Millionen Issues [17]. In Kombination mit einem Alexa-Rang, welche die Popularität einer Internetseite relativ zu anderen weltweit einstuft,

von 90 [18] stellt sich Github als potentester Versionsverwaltungsdienst für die gewünschten Daten heraus. Um an die Metainformationen zu kommen, wird die offizielle Github Anwendungs-Programmier-Schnittstelle (Github API) verwendet, um Entwicklern den Zugriff auf Github auf Programmierebene zu ermöglichen.

Um nun aus der gewaltigen Masse an Issues vereinzelt akkurate Ergebnisse geliefert zu bekommen, wird unter anderem die „Google Custom Search JSON API"(Google API) als Filter verwendet. Das ist eine von Google bereitgestellte Schnittstelle für Entwickler, die den Zugang zur gleichnamigen Suchmaschine auf einer Programmierebene ermöglicht [19]. Sie wird in dieser Arbeit in Verbindung mit einer „Google Programmable Search Engine" (Google Search Engine) genutzt, welche eine selbst erstellte Instanz dieser Suchmaschine ist [20]. Die Suchanfrage, welche beim Verwenden der Google API noch feiner eingestellt werden kann, ist so gestellt, dass Resultate nur Issues aus Github liefern. Über festgelegte Schlüsselwörter wird bei der Suche auch gleichzeitig der Inhalt jedes Issues geprüft. Dabei werden Diskussionen in kürzester Zeit auf händisch untersuchbare Mengen reduziert. Zudem werden diese Resultate gleichzeitig durch einen Algorithmus von Google auf ihre Relevanz sortiert.

Die gefilterten Suchergebnisse werden daraufhin mit relevanten Metainformationen in eine Datenbank gespeichert. Hierzu wird eine SQLite-Datenbank verwendet. Diese ist nicht von einem externen Server abhängig und wird deswegen lokal aufgesetzt. Zudem funktioniert sie auf allen Betriebssystemen und bietet alle Funktionalitäten, die in dieser Arbeit benötigt werden, wodurch kein Bedarf an einer anderen und komplexeren Datenbankbibliothek herrscht [21]. Die Datenbank wird zum einen für das Caching verwendet. Es ist eine Methode, die das erneute und zeitaufwendige Abfragen von bereits getätigten Einträgen verhindert, indem die bereits gespeicherten Einträge verwendet werden. Zum anderen aber auch, um langfristig gesehen einen eigens sortierten Datensatz zu erstellen, mit dem der Anwender effizienter arbeiten kann.

Während der Suche werden die angefragten Issues dem Anwender aufbereitet ausgeben. Dabei wird sich für die Analyse auf relevante Metainformationen beschränkt. Darunter auch ein direkter Link zum Issue und eine Beschreibung des dazugehörigen Repositories, um den Nutzer die Zugänglichkeit zum Inhalt und Kontext zu erleichtern. Dies geschieht in einem interaktiven Modus, siehe 1, über welchen man mittels

Eingaben in der Konsole mit dem Programm kommuniziert und dabei weitere Methoden ausführt.

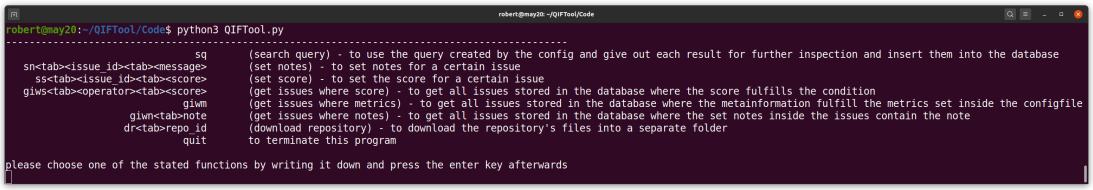

Abbildung 1: Interaktiver Modus bei der Ausführung des QIFTools.

3.2 OSCTool

Damit die Gedankengänge des folgenden Lösungsansatzes nachvollziehbar sind, wird in diesem Textabschnitt das anfängliche Verständnis der Hauptproblematik benannt. Diese ist nicht zu weit von der eigentlichen entfernt und befasst sich ebenfalls mit technischen Schulden. Der Unterschied liegt darin, dass zunächst allgemein ein Bedarf an Repositories bestand, anstelle von Diskussionen aus Issues. Dadurch wurden noch weitere Metriken abgespeichert, die sich mehr auf Repositories als Issues beziehen. Außerdem war angedacht, dass dem Anwender lediglich die Metriken alle bereitgestellt werden. Dabei sollte sich der Anwender damit auseinandersetzen, welche Metriken wie eingestellt am ehesten zum Fund dieser gewünschten Daten führen.

Der anfängliche Lösungsansatz verwendet ebenfalls den Datensatz von Github und basiert auf derselben Begründung. Über die Github API wird der Zugriff auf weitestgehend zufällige Repositories ermöglicht. Schon vorab können diese auf ihre Speichergröße vorgefiltert werden. Diese werden iterativ auf ihre Qualität untersucht, indem zum einen alle relevanten Metainformationen extrahiert werden und zum anderen drei Kommunikationsschnitstellen individuell betrachtet werden. Darunter gehören Issues, „Pull Requests“, welches Issues mit einer festgelegten Thematik sind, und „Commits“. Commits stellen dabei geteilte Änderungen einer Datei innerhalb eines Projekts dar und können ebenfalls von Entwicklern kommentiert werden. Dabei wird jeder einzelne Kommentar der genannten Kommunikationsschnittstellen auf die Verwendung von vorab festgelegten Schlüsselwörtern geprüft.

Das Speichern der Ergebnisse jedes einzelnen Repositories und Issues, Pull Requests und Commits wird ebenfalls in einer SQLite-Datenbank realisiert. Hier Bedarf

es sich an gleicher Begründung wie in 3.1 bereits erörtert. Auf lange Sicht gesehen sollte dabei eine gigantische Datenbank entstehen, die aus Cachinggründen jedes Repository mit abspeichert, obgleich es relevant für technische Schulden ist oder nicht. Die Teilmenge an relevanten Repositories sollte dabei jedoch den Fokus auf die zugeschnittenen Schlüsselwörter haben und dadurch nur ein Bruchteil so groß sein.

Anschließend sollte der Anwender sich mithilfe von SQL-Befehlen und den bereitgestellten Metriken die gewünschten Daten ausgeben lassen können.

3.3 Alternativen

Zu Beginn der Arbeit existierte lediglich eine vage Vorstellung, wie die Software zu realisieren war. Dadurch ergab die frühzeitige Recherche nach bereits existierender Software keine Treffer. Erst nach einiger Zeit entwickelte sich der OSCTool-Lösungsansatz auf natürliche Weise in die Richtung einer Datenbank, die nicht nur im Nachhinein die Ergebnisse abspeichert, sondern ein lokales Abbild einer Teilmenge von Github erstellt. Daher ergab sich anfänglich lediglich ein Ansatz, der bis weiteres auch verfolgt wurde.

Während der erwähnten Zwischenpräsentation sind nach näherer Erläuterung der Wunschergebnisse nun einige Alternativansätze hervorgekommen. Dazu zählen das in 2.2 erwähnte Projekt Software Heritage, der Ansatz über die Google-Suchanfrage und das GHTorrent-Projekt, welches sich als sehr ähnliches Endprodukt zu dem OSCTool-Ansatz herausstellte.

3.4 Abwägung der Alternativen

Um sich im weiteren Verlauf der Arbeit auf einen Ansatz zu fokussieren, wurden der OSCTool-Ansatz und die Alternativansätze bewertet. Aus Zeitgründen wurden die Ansätze nur so weit untersucht, bis eine Entscheidung getroffen werden konnte, die am effektivsten erschien.

3.4.1 OSCTool

Das Tool ist zum Zeitpunkt der Tests so weit geschrieben worden, dass ein authentischer Durchlauf ausgeführt werden konnte. Es wurde über zwölf Stunden laufen gelassen und mit fünf Schlüsselwörtern initialisiert. Aufgrund der aufwendigen Schlei-

fen, die mit jedem zusätzlichen Schlüsselwort einhergehen, enthielt die Datenbank im Anschluss rund 1000 Einträge. Für die relative Bewertung der Ansätze und Durchläufe untereinander wurde ein eigens erstelltes Rangsystem verwendet. Den Resultaten wurden jeweils je nach Effektivitätsgrad einer von drei unterschiedlichen Rängen zugewiesen. Diese sind mittels der eigenen Bewertung vergeben worden und nicht über eine programmierte Kalkulation. Mit „none“ wurden jene gekennzeichnet, die zwar die genannten Schlüsselwörter beinhalten und diese auch im richtigen Kontext verwendet haben, aber keine eigentliche Korrelation zu Inhalten bezüglich Technical Debt besitzen. Den Rang „low“ wurden Resultaten zugewiesen, die im Grunde die Intention und die Konzepte von Technical Debt miteinander kommuniziert haben, aber noch keine Diskussion geschaffen haben, die sich mit der Thematik auseinandersetzt. Als hochwertig zählen Resultate mit der Kennzeichnung „high“ welche Diskussionen zum Thema Technical Debt besitzen und sich inhaltlich mit dessen Konzepten und Ansätzen zur Bewältigung von syntaktischer Umgestaltung befassen. Die Ergebnisse der Durchläufe sind in Tabellen erfasst. Die erste Spalte bezieht sich auf die Position innerhalb der Trefferliste. Die zweite Spalte weist den Resultaten durchlaufübergreifend einen Identifikator zu, während die letzte Spalte die eigens vergebenen Ränge beinhaltet. Für die Durchläufe wurde sich auf die ersten zehn Treffer beschränkt. Mittels SQL-Befehl wurden die Einträge der Datenbank sowohl nach den Schlüsselwörtern „technical debt“ oder „refactor“ als auch über Metriken gefiltert. Bei den Metriken wurde auf eine Anzahl an Entwickler von mehr als fünf und einer Kommentaranzahl der Issues von mehr als zehn gesetzt.

Die Einträge aus Tabelle 1 haben überwiegend einen Rang von „none“ und einige einen von „low“ und einem Ergebnis, das sich inhaltlich zwischen „low“ und „high“ befindet. Daraufhin wurde der SQL-Befehl nochmals abgeändert ausgeführt, indem nur Einträge mit dem Schlüsselwort „technical debt“ und einer Entwickleranzahl von mehr als fünf angezeigt wurden.

Die Menge an Resultaten aus Tabelle 2 belief sich hierbei auf lediglich drei Einträge, wobei zwei davon einen Rang von „high“ und einen von „low“ besitzen. Aufgrund der wenigen Einträge in der Datenbank und der daraus resultierenden geringen Menge an post-selektierten Ergebnissen, kann die Effektivität des Tools im Ganzen nicht eindeutig eingestuft werden. Das Filtern nach konkreten Schlüsselwörtern, die dicht

result	id	ranking
1	28	low
2	29	none
3	30	low
4	31	none
5	32	low - high
6	33	none
7	34	none
8	35	none
9	36	low
10	37	none

Tabelle 1: Ergebnisse der Anfrage vom OSCTool. Mit zwei Schlüsselwörtern und zwei Metriken, Quelle: Eigene Daten

result	id	ranking
1	38	low
2	39	high
3	40	high

Tabelle 2: Ergebnisse der Anfrage vom OSCTool. Mit einem Schlüsselwort und einer Metrik, Quelle: Eigene Daten

an der Thematik sind, weist nach der zweiten Testiteration ein mögliches Potential auf.

3.4.2 Software Heritage

Das Projekt bot auf den ersten Blick eine vielversprechende Ausgangssituation. Die große Datensatzmenge in Kombination mit einem eingebauten Suchfeld ermöglicht die sofortige Eingabe von Schlüsselwörtern. Zusätzlich werden die Repositories inhaltlich nach den Schlüsselwörtern gefiltert. Die Ergebnisse waren jedoch nicht hochwertig. Angezeigt wurden lediglich die Verweise auf die Repositories als ganzes und ohne Bezug darauf, auf welche Metainformationen innerhalb des Projektes sich die in das Suchfeld eingegebenen Schlüsselwörter beziehen. Aufgrund des fehlenden Be-

zugs, der nicht weiter einstellbaren Filter und der Tatsache, dass einem keine Diskussionen, sondern nur das Projekt geliefert werden, erweist sich Software Heritage als ungeeignet für den weiteren Verlauf als Lösungsansatz.

3.4.3 GHTorrent

Im Grunde gleicht das Fundament hier dem vom OSCTool. Der entscheidende Unterschied hierbei ist, dass die Filterung nach Schlüsselwörtern noch nicht stattgefunden hat. Das Abfragen von Issues würde schneller ausfallen, da nicht erst auf die Repositories, sondern direkt auf die Issue-Tabellen zugegriffen werden kann. Das zu erreichende Potential ist hierbei mit dem vom OSCTool-Ansatz gleichzusetzen, da der Ausgangsdatensatz und Vorgang zur Lösung identisch sind. Es gibt jedoch auch Nachteile, die mit der Verwendung dieses Ansatzes zusammen kommen. Zum einen schränkt das Rate-Limit der Github-API einen nicht ein, jedoch besitzt die Verwendung der externen Datenbank seine eigenen Einschränkungen. Dabei ist unter anderem einem nur eine begrenzte Befehlslaufzeit bereitgestellt. Zudem muss auf eine Rückmeldung von GHTorrent gewartet werden, um auf die Datenbank zugreifen zu können [22]. Um diesen Einschänkungen zu entgehen kann das GHTorrentabbild herunterzuladen werden. Für diese Momentaufnahme des Umfangs von Github wird rund 100GB an Speicherplatz benötigt [23], was das Tool letztlich als unflexibel gestaltet . Zum anderen besitzt das Projekt wie in 2.2 erwähnt keinen vollständigen Datensatz. Im direkten Vergleich zum OSCTool könnte GHTorrent jedoch als Fundament dienen, um den langsameren Zugriff über die Github-API an Metainformationen zu entgehen. Das Fehlen einiger Repositories vor 2012 wäre aufgrund der hinreichenden Menge an Issues noch argumentativ zu rechtfertigen. Aus Speicherplatz- und Zeitgründen wurden keine manuelle Tests durchgeführt.

3.4.4 Google Suchanfrage

Für die Tests der Google-Suchanfrage ist hierbei auf manuelles Eingeben einiger Suchanfragen gesetzt worden, die so auch sonst automatisiert an die Suchmaschine weitergegeben werden würden. Die Anfragen sind dabei so gestaltet, dass sie einerseits nur auf der Github-Webseite suchen. Andererseits sind sie noch durch die Verwendung des Wortes „issue“ im URL-Attribut so eingeschränkt, dass nur nach

Issues von Github gefiltert werden. Des Weiteren wird zugleich der Inhalt auf die Schlüsselwörter über Google geprüft, welche eine unvergleichliche Effizienz liefert. Innerhalb weniger Sekunden werden hierbei voraussichtlich gewünschte Datensätze ausgegeben. Für die Verwendung von Google sind einige Kombinationen an Schlüsselwörtern ausgetestet worden. Für die Auswertung der Ergebnisse wurde auch hier das in 3.4.1 beschriebene Rangsystem verwendet und auf die ersten zehn Resultate beschränkt. Angefangen mit vier Schlüsselwörtern, die konjunktiv verknüpft sind, ergab sich die Resultatmenge auf 26 Issues.

result	id	ranking
1	1	high
2	2	high
3	3	high
4	4	low
5	5	low
6	6	low
7	7	low
8	8	low
9	9	none
10	10	high

Tabelle 3: Ergebnisse der Anfrage über die Google API. Mit vier Schlüsselwörtern, Quelle: Eigene Tabelle

Vier der zehn Ergebnisse aus Tabelle 3 erweisen ein „high“ und fünf der restlichen ein „low“. Bei der Verwendung von drei verundeteten Schlüsselwörtern ergaben sich zu dem Zeitpunkt 121 Resultate. Aus Tabelle 4 sind sechs als „high“ und vier als „low“ eingestuft.
Die dritte Iteration verwendete zwei konjunktive Schlüsselwörter mit einer Ergebnismenge von 809 Treffern. Zwei der Ergebnisse aus Tabelle 5 wurden mit „high“und sechs davon mit „low“ eingestuft. Die manuellen Tests zeigten, dass sich mit der richtigen Einstellung der Anfrage die Ergebnisse der Google-Suchanfrage als potent erweisen können.

result	id	ranking
1	1	high
2	11	high
3	12	low
4	13	low
5	2	high
6	14	high
7	15	low
8	16	high
9	17	low
10	3	high

Tabelle 4: Ergebnisse der Anfrage über die Google API. Mit drei Schlüsselwörtern, Quelle: Eigene Daten

result	id	ranking
1	18	none
2	19	none
3	20	low
4	21	low
5	22	low
6	23	low
7	24	low
8	25	low
9	26	high
10	27	high

Tabelle 5: Ergebnisse der Anfrage über die Google API. Mit zwei Schlüsselwörtern, Quelle: Eigene Daten

3.4.5 Auswertung der Abwägungen

Teile der Bewertung aus den Tests der Google-Suchanfrage wurden ebenfalls mit Herrn Brekenfeld, einem in der Arbeitsgruppe tätigen Mitarbeiter, abgeglichen. Sein

Fachwissen und seine wertvolle Meinung als Spezialist sollten hierbei die Effektivität der Testfunde bestätigen. Der direkte Vergleich der Tabelle 4 mit Herrn Brekenfelds Vergabe in Tabelle 6 zeigt, dass die meisten Einstufungen seinerseits mit der eigenen Rangvergabe übereinstimmen, sodass hier die Gewissheit gegenüber der Effektivität und den korrekten Datensätzen als gewünschtes Endprodukt besteht.

result	id	ranking	ranking_herr_brekenfeld
1	1	high	high
2	11	high	medium - high
3	12	low	low
4	13	low	low
5	2	high	high
6	14	high	medium
7	15	low	low
8	16	high	low
9	17	low	none
10	3	high	high

Tabelle 6: Vergleich der eingeschätzten Werte von Herrn Brekenfeld Tabelle 4. Mit drei Metriken, Quelle: Eigene Daten

Zum einen bietet der Ansatz von Google den identischen Datensatz von Github wie das OSCTool und auch GHTorrent an und zum anderen liegt die Leistungsfähigkeit der Schlüsselwortfilterung weit über der der Alternativen. Zusätzlich wird vom Relevanzalgorithmus von Google profitiert, um eine verbesserte Sortierung der Treffer zu erhalten. Bei den manuellen Tests zeigte sich, dass Issues, die mit einem „high“ versehen sind, tendenziell auch viele Kommentare besitzen. Demnach ist die Kommentaranzahl eine zusätzliche Metrik, mit der der Effektivitätsgrad weiter erhöht werden kann und zeigt somit mögliches Ausbaupotential. Das Filtern von Schlüsselwörtern hat sich als potenteste Metrik im Vergleich zu anderen Metainformationen erwiesen, sodass weitere unterstützende Metriken in Verbindung mit Schlüsselwörtern verwendet werden. Aufgrund der aufgeführten Argumente und der Tatsache, dass Software Heritage ungeeignet ist, viel die Entscheidung auf die Google-Suchanfrage.

4 Implementierung des Tools

4.1 Bibliotheken

Das OSCTool verwendet für den gleichen Zweck dieselben Bibliotheken wie das QIF-Tool. In diesem Abschnitt werden daher die Bibliotheken für beide Ansätze betrachtet. Im Zusatz dazu verwendet das QIFTool jedoch noch die Google API, welche alleinstehend für den neuen Ansatz zu betrachten ist.

„**PyGithub**“ist das für Python verwendete Modul der offiziellen „Github REST API v3“. Es erlaubt den Zugriff auf die Metainformationen des Githubdatensatzes und wird in der Arbeit genutzt, um unter anderem Metriken für die Datenbank zu extrahieren.

Das Modul „**sqlite3**“ermöglicht die Verwendung der in 3.1 genannten SQLite Datenbank. Zudem bildet sie eine nach DB-API 2.0 SQL-konforme Schnittstelle, da Python standardmäßig nicht fähig ist mit relationalen Datenbanken zu kommunizieren [24].

Die „**configparser**“Bibliothek ermöglicht die Erzeugung und das Auslesen einer Konfigurationsdatei. Diese wird vom Endnutzer verwendet, damit dieser mit dem Programm kommunizieren kann, ohne sich in den Quelltext einfinden zu müssen. Zudem bietet diese Struktur einen eleganteren Umgang sowohl für den Anwender als auch vom Entwickler.

Beim Erreichen des temporär limitierten Zugriffs (Rate-Limit) auf die Github API wird das „**time**“-Modul genutzt, damit das Programm auf die nächste Zurücksetzung des Rate-Limits wartet.

Für die Berechnung der zu wartenden Zeit findet die Bibliothek „**datetime**“ihren Nutzen.

Das Modul „**sys**“ermöglicht die Verwendung von systemspezifischen Parametern und Funktionen. Hauptsächlich wird es genutzt, um eine detailliertere Fehlermeldung auszugeben. Zudem wird damit das Programm im Falle einer Ersterzeugung der Konfigurationsdatei aktiv beendet.

Die Schnittstelle, um mit dem Betriebssystem der lokalen Maschine zu kommunizieren, ermöglicht die Bibliothek „**os**“. Hierbei werden Ordner erstellt, Pfade gesetzt und Dateien sowohl erzeugt als auch auf ihre Existenz geprüft.

Die „**googleclientapiclient**“-Bibliothek stellt die in 3.1 erläuterte Schnittstelle zur Ver-

wendung der Google Suchanfrage dar. Sie stellt die Verbindung zur Suchmaschine her und speist ihr auch gleichzeitig die Query ein, mit der gesucht werden soll.

4.2 Erfolgte Umsetzung

4.2.1 OSCTool

Der in 3.2 beschriebene Lösungsansatz ist bis hin zur Speicherung der Daten in die Datenbank realisiert worden. Dazu zählen das vollautomatisierte Auswerten von randomisierten Repositories, das Auslesen der Konfigurationsdatei auf individuell angepasste Schlüsselwörter und Pfade sowie die benutzerfreundliche Konsolenausgabe, welche dem Endnutzer über den momentanen Stand des Programms informiert. Ein „Downloader“, der die gewünschten Repositories identisch zu ihrer Ordnerstruktur auf Github herunterlädt ist ebenfalls umgesetzt. Aufgrund des in 1.2 erläuterten Wechsels sind folgende Funktionalitäten nur konzipiert verblieben. Der zuvor erwähnte Downloader ist zwar implementiert, jedoch fehlt noch die automatisierte Einspeisung von mehreren Repository-Identifikatoren. Benutzerfreundliche Masken, welche die Einträge aus der Datenbank über SQL-Befehle ausgeben, sind nicht umgesetzt worden. Diese sollen dem Anwender dabei helfen sich nicht mit der SQL-Syntax und dem Zugriff auf die lokale Datenbank auseinandersetzen setzen zu müssen. Ein interaktiver Modus, in welchem der Endnutzer über Konsoleneingabe mit dem Programm kommunizieren kann, fehlt. In diesem Modus wäre der Nutzer in der Lage, die genannten Masken sequentiell auszuführen, Repositories herunterzuladen oder auch den allgemeinen Suchvorgang zu initialisieren. Außerdem ist hier mit der Idee gespielt worden, das Programm über mehrere „Threads“, welches simultane Instanzen des Programms zur Laufzeit sind, laufen zu lassen. Das Verwenden von Threads ermöglicht eine effizientere Nutzung der limitierten Zugriffe, da das Tool mit der momentanen Implementierung nicht an das Rate-Limit stößt.

4.2.2 QIFTool

Für diesen Ansatz ist alles konzeptionelle in 3.1 vollständig umgesetzt. Folgende kleinere Funktionalitäten sind jedoch nicht umgesetzt. Der in 4.2.1 erwähnte Downloader für Repositories ist eingebaut, jedoch fehlt auch hier die automatisierte Umsetzung. Für die erwünschten neuen Datensätze ist jedoch kein Bedarf an Repositories, sodass

der Kern für die Umsetzung des automatisierten Downloaders für mögliche Erweiterungen im Quelltext verweilt. Einen direkten Verweis bei der Ausgabe auf Issues, die innerhalb eines Issues auf andere verweisen, ist ebenfalls nur konzeptionell im Quelltext vorhanden. Bei näherer Auseinandersetzung mit dieser Funktionalität ist ein Hindernis aufgetreten, das die tatsächliche Realisierung als zu Zeitaufwendig für den momentanen Stand der Arbeit gestaltet.

4.3 Qualitätssicherung

Einer der zu prüfenden Aspekte ist die Robustheit des Tools. Sie beschreibt die Eigenschaft unter ungünstigen Bedingungen noch zuverlässig zu funktionieren [25]. Zur Prüfung der Robustheit sind Tests durchgeführt worden, die sich mit Grenzfällen befassen. Dazu gehören Situationen wie das Erreichen des Rate-limits der Github API, Überiterieren von Suchtreffern der Google-Suchanfrage und redundante Eintragen eines Treffers in die Datenbank. Für die Tests wurden Ausgangssituationen erstellt, die genau diese Grenzfälle jeweils ausgetestet und die implementierte Problembehandlung geprüft haben. Beispielsweise ist zu dem Zeitpunkt des Tests die Syntax des Quelltexts so umgesetzt gewesen, dass es über eine ineffiziente Schleife zum Erreichen des Rate-limits kam. Über manuelle Beobachtung, wurde dann bestätigt, dass das Tool korrekt nach Konzept handelt. Für die Gewährleistung der Korrektheit der gespeicherten Daten sind manuelle Einsichten vorgenommen worden. Dazu sind nach Durchläufen von Datenbankeinträgen die Metainformationen in den einzelnen Spalten mit den entsprechenden Metainformationen auf Github selber abgeglichen worden. Zur Absicherung an gebotener Funktionalität sind manuelle Tests durchgeführt worden. Zum einen ist jede Methode, die in dem interaktiven Modus aus 4.2.2 genutzt werden kann, auf ihre Wirksamkeit geprüft worden und zum anderen sind die individuellen Attribute der Konfigurationsdatei ebenfalls auf ihre Funktionalität getestet worden.

Aufgrund der Menge an manuellen Tests und der Verwendung von hauptsächlich externen Bibliotheken sind für dieses Tool keine automatisierten Tests geschrieben worden. Diese Bibliotheken sind nämlich in sich geschlossen bereits ausführlich getestete Module, sodass der eigenen Einschätzung nach eine Verknüpfung dieser keine weiteren Testansätze bedarf.

4.4 Reifegrad

Im folgenden Abschnitt wird die persönliche Stellungnahme zum Tool und Quelltext aus einer Entwicklerperspektive genommen.

4.4.1 Entwurf

Das Tool erfüllt drei Hauptaspekte, die miteinander verflochten dem Entwurf seine Qualität zusprechen. Eine davon ist die Kernfunktionalität und beschreibt die Umsetzung der Vorgehensweise im Bezug auf ihre Effizienz. Der zweite Aspekt ist die Aussicht auf die langfristige Effektivität des Tools. Damit ist die Datenbank in Hinblick auf einen immer wertvoller werdenden Datensatz gemeint, sodass es sich nicht lediglich um Caching handelt. Dies gelingt durch die Verwendung von Spalten, die vom Anwender eingetragen worden sind. Diese stufen das untersuchte Issue relativ zu anderen ein. Der letzte Aspekt befasst sich mit dem Wachstum des Tools. Damit ist die Konfigurationsdatei gemeint, mit der das Tool mithilfe empirischer Eingaben von Metriken in der Lage ist, hochwertigere Treffer zu erzielen, als es das zu anfangs tut. Außerdem wird durch die Konfigurationsdatei und dem interaktiven Modus für eine klare Trennung zwischen Front- und Back-End gesorgt. Damit sind zwei Schichten in der Softwareentwicklung gemeint, bei der die benutzerfreundliche Präsentationsschicht von der komplexeren Datenzugriffsschicht getrennt ist, um dem Nutzer die Anwendung zu erleichtern.

4.4.2 Programmierstil

Das Tool beinhaltet die Verwendung von selbst erstellten Klassen, das Abfangen und den Umgang von Fehlermeldungen, das Arbeiten mit Dateien bezüglich dessen Zugriff und Erstellung und die Einbindung von weiteren Bibliotheken. Syntaktisch gesehen, ist der Quelltext stets an einer „best-practice“-Schreibweise angelehnt, was diesen leserfreundlich und übersichtlich macht. Zudem ist jede Funktion mit einem Funktionskommentar versehen, um den Einstieg in die Semantik des Tools zu erleichtern. Den aufgeführten Punkten nach ist der Programmierstil meiner Einschätzung nach auf einem fortgeschrittenen Niveau zu betrachten.

4.4.3 Verlässlichkeit

Aufgrund der Tests, die in 4.3 durchgeführt worden sind, und der Fehlermeldungsbehandlung ist das Tool der eigenen Einschätzung nach zuverlässig. Lediglich hypothetische große Umstrukturierungen, die toolexterne Angelegenheiten betreffen, sind mögliche Fehlerpunkte für die Verlässlichkeit des Tools.

4.4.4 Benutzbarkeit

Das Tool bietet aufgrund seiner in 3.4.5 bestätigten Qualität an Funden und der Bereitstellung der gewünschten Anforderungen eine Benutzbarkeit im Anwendungsbereich. Zusätzlich sind eigens eingeflossene Funktionalitäten, die zur Effektivität des Tools beitragen sollen, und eine benutzerfreundliche Ausgabe implementiert worden.

5 Problembewältigung

In diesem Abschnitt werden Bewältigungen von aufgetretenen Problemen angesprochen, die für das Tool einen besonderen Mehrwert darstellen.

5.1 Rate-Limit

Eine essentielle Qualitätssicherung stellt die in 4.3 genannte Behandlung des Rate-Limits dar. Auch wenn die Anzahl an Anfragen im jetzigen Zustand nicht an das aktuelle Rate-Limit von Github stößt, sollte auch aus unberechenbaren Gründen, eine anständige Behandlung dieser Ausnahmen vorgenommen sein. Dazu wären zwei Ansätze möglich gewesen. Einerseits könnte das Programm in regelmäßigen Abständen auf die Github API zugreifen, um einen kontinuierlichen Fluss an bearbeiteten Daten zu bekommen. Andererseits könnte das Tool bis an die Grenze des Rate-Limits zugreifen und anschließend bis zur nächsten Zurücksetzung des Rate-Limits warten. Letztendlich ist die Entscheidung auf die zweite Variante gefallen, da es sich hierbei um den effizienteren Ansatz handelt. Das Programm muss sich nämlich während der Kernfunktionalität nicht noch nebenbei mit der Berechnung der relativen Abständen bemühen. Ließe man diese präzise Vorgehensweise weg und setzt eher auf einen absoluten Abstand, so kommt es aufgrund von Hardware- und Laufzeitunterschieden

höchstens zu ungenauen Werten. Bei der Berechnung der Wartezeit auf die Zurücksetzung musste zudem noch darauf geachtet werden, dass es sich hierbei um möglicherweise unterschiedliche Zeitzonen handelt. Bei der Zeitangabe von Github handelt es sich um die Zeitzone der Koordinierten Universellen Zeit (UTC), sodass hier eine Umwandlung der lokalen Zeit, welche sich auf die der laufenden Maschine bezieht, in die UTC-Zeitzone vorgenommen wurde. Dadurch konnte die korrekte Differenz an Sekunden berechnet werden.

5.2 Google Suchanfrage

Wie in 3.4.4 erläutert nutzt die Google-Suchanfrage die URL, um allgemein an Issues zu gelangen. Ein Präfix dieser URLs kann Suchtreffer bilden, bei denen es sich nicht um ein konkretes Issue handelt. Für die Bildung von eindeutigen Issues wird die Issuenummer in der URL benötigt. Der Pool an Suchtreffern besteht somit aus konkreten Issues oder ungewollten Issueübersichten der Repositories. Die Suchanfrage gibt stets maximal 100 Ergebnisse zurück, sodass die ungewollten Treffer nicht nur ignoriert werden, sondern gänzlich aus der Ergebnisliste vorgefiltert werden sollten. Dadurch sollte eine Maximierung der Wahrscheinlichkeit an hochwertigen Treffern entstehen. Dazu wurde zunächst versucht das „inurl“-Attribut mit einem zusätzlichen „/“ zu erweitern, um auf konkrete Seiten mit Issues zu verweisen. Dieses würde sich auf die Syntax der URL beziehen und nur übereinstimmende Treffer zulassen. Die Semantik der Google API ignoriert jedoch jede Art von Symbolen, sodass der ungewollte Pool an Treffern identisch blieb. Daraufhin wurde in der Google API selber nach einer möglichen Lösung gesucht. Es zeigte sich ein Attribut innerhalb der zu übergebenden Werte an die Suchmaschine. Das Attribut bekommt eine Zeichenkette und gibt nur Treffer aus, die die Zeichenkette exakt inhaltlich im Quelltext enthalten. Somit musste nur noch lediglich eine Zeichenkette gewählt werden, die in jedem konkreten Issue vorkommt, jedoch niemals in der Issueübersicht.

6 Durchführung

In diesem Abschnitt wird chronologisch der Werdegang dieser Arbeit angesprochen. Dabei wird es sich erneut um eine Aufteilung zwischen anfänglichem und neuem

Ansatz handeln.

6.1 Durchführung - OSCTool

Die *ersten zwei Wochen* fingen mit der Einrichtung aller nötigen Entwicklerumgebungen und Tools an. Dazu gehörten Tools wie Overleaf, einem kollaborativen und cloudbasierten LaTeX-Editor. Ein eigenes privates Github-Repository wurde eingerichtet, um hauptsächlich den Fortschritt der Arbeit stets gesichert zu haben. Trello wurde verwendet, um sich eine strukturierte und übersichtliche To-do-Liste anzufertigen, die einem stets den Stand und Fortschritt visualisiert. Zudem sind bereits einige Anforderungen gestellt worden, sodass es bereits einen roten Faden gab, an dem sich orientiert werden konnte. Inhaltlich wurde demnach mit der Recherche nach möglichen Metriken angefangen, die auf den ersten Blick als potentiell wertvoll betrachtet wurden. Diese wurden anschließend näher betrachtet, auf ihre Metainformationen geprüft und gegebenenfalls aussortiert, um auf einen gewissen Satz an Metriken zu kommen, die für die Thematik effektiv erscheinen. Außerdem wurde sich mit dem Rate-Limit befasst, wie sich dieses zusammensetzt und welche Zugriffe wie viele Anfragen bilden.

In der *dritten Woche* wurde sich mit der Anfrage befasst, die noch vor dem Filtern der Metriken zum Einsatz kommt. Für diese Art von Filterung gibt es über ein Dutzend sogenannte Qualifiers. Das sind Attribute, die Repositories erfüllen müssen, um für die sonst zufällige Auswahl erfasst zu werden. Lediglich ein Qualifier war jedoch von Nutzen, um vorab den Datensatz sinnvoll zu reduzieren. Ferner ist außerdem nach der Methode des erwünschten Cachings diskutiert worden. Hierbei kamen Ansätze wie die Verwendung einer simplen Textdatei oder auch CSV-Datei zur Sprache. Letztendlich fiel nach einigen Abwägungen die Entscheidung auf die SQLite-Datenbank.

Die Realisierung der SQLite-Datenbank in ihrer ersten Form fand in der *vierten Woche* samt Erweiterungen von Spalten, die nach und nach dazu kamen, statt. Zusätzlich wurde hier eine ebenfalls frühe Version des Downloaders für die Repositories umgesetzt. Außerdem wurde gegen Ende der Woche die Robustheit, wie in 5.1 erläutert, für alle Metriken gegenüber dem Rate-Limit implementiert.

Für die *fünfte Woche* sollte anfänglich die Konfigurationsdatei durch eine simple

Textdatei umgesetzt werden, jedoch fiel nach Recherche die Entscheidung auf die in 4.1 erwähnte Bibliothek, die eine wesentlich elegantere Realisierung einer Konfigurationsdatei darstellt. Für die Datenbank wurde außerdem die Hauptstruktur gebildet, indem für einige Metriken individuelle Tabellen erstellt wurden. Des Weiteren wurde das Tool so weit geschrieben, dass es nun über die Verknüpfungen von Konfigurationsdatei und der Datenbank authentische Testdurchläufe führen konnte.

Mit dem derzeitigen Stand begann die *sechste Woche* mit einigen Testdurchläufen wie in 4.3 beschrieben. Hierbei sind auch Ansätze über die Verwendung von Threads, wie in 4.2.1 erwähnt, hervorgekommen. Diese sollten den ineffizienten Ergebnissen der Testdurchläufe entgegenwirken. Nach mehr Untersuchungen im eigenen Code stellte sich heraus, dass durch eine Umstrukturierung der Syntax die Menge an Zugriffen über die API so weit gesenkt werden konnte, dass die Umsetzung von Threads nicht mehr erforderlich war. Bedenken über das Erreichen des Reihenlimits der Datenbank wurden nach Recherche widerlegt, da bereits vorher die maximale Speichergröße der Datenbank selber erreicht werden würde [26]. Des Weiteren wurden Fehlerbehebungen bezüglich der Einträge und dem Einfügen dieser vorgenommen.

Am Anfang der *siebten Woche* wurden die Folien für die aus 1.2 erwähnte Zwischenpräsentation erstellt.

6.2 Durchführung - QIFTool

Ein Termin direkt nach der Zwischenpräsentation aus 1.2 wurde mit dem Betreuer gehalten, um über den weiteren Verlauf zu sprechen. Daraufhin ergaben sich einige Anhaltspunkte, mit denen sich ein klares Ziel erarbeiten ließ. Es ereignete sich das in 3.4 erörterte Vorgehen der Alternativen, sodass der Fokus des neuen Ansatzes der Arbeit fest stand. Für ein besseres Verständnis sind auch hier einige wissenschaftliche Ausarbeitungen zu ähnlichen Arbeiten gelesen worden.

Die *Woche acht* begann mit der Umsetzung des neuen Ansatzes. Zuallererst wurde die Methode implementiert, der sich mit der Google API und ihren Anfragen befasst. Dabei zeigte sich das in 5.2 erörterte Problem der ungewollten Suchergebnisse. Diese sind wie beschrieben durch die Verwendung eines zusätzlichen Attributes in der Suchanfrage vorab herausgefiltert worden. Außerdem wurde viel Zeit darin investiert sich in die Ergebnisse der Google Search JSON API einzufinden und die essentiellen

Attribute herauszufiltern. Ferner wurde nun nebenbei auch die schriftliche Ausarbeitung dieser Arbeit angefangen.

Der neue Entwurf ähnelt dem vorigen in seinem Aufbau. Für die restlichen Funktionalitäten konnte daher in der *neunten Woche* Quelltext des OSCTools übernommen werden. Dazu zählen Abschnitte wie die SQLite-Datenbank, Konfigurationsdatei und Github API. Das Wiederverwenden sparte nicht nur reine Programmierzeit, sondern zusätzlich noch Einarbeitungszeit bezüglich der verwendeten Bibliotheken. Auch wurde einiges an Zeit für die schriftliche Ausarbeitung aufgebracht. Während der regelmäßigeren Termine mit dem Betreuer wurde zudem inhaltlich mehr organisatorisches zur kommenden Abgabe besprochen.

In der *zehnten Woche* wurden um das erstellte Grundgerüst die weiteren Funktionalitäten umgesetzt. Dabei handelt es sich insbesondere um den interaktiven Modus aus 3.1, welcher die Benutzerfreundlichkeit und Bedienbarkeit des Tools realisiert. Dieser wurde auf die Bedürfnisse von Herrn Brekenfeld zugeschnitten und besitzt somit die essentiellen Funktionalitäten, um effektiv im Forschungsbereich angewendet zu werden. Herr Brekenfeld hat das Tool im Nachhinein getestet und gab daraufhin eine positive Rückmeldung bezüglich der gewünschten Ausgabe. Mit dem Annähern der Fertigstellung des Tools verschob sich die Zeitaufteilung mit einem wesentlich größerem Anteil auf die Ausarbeitung.

Für das Tool wurden in der *elften und letzten Woche* noch einige letzte aufgeschobene Funktionalitäten umgesetzt. Für das weitere Schreiben der Arbeit konnte allgemein ein großer Vorteil aus dem geführten Tagebuch der Arbeit gezogen werden, welches besonders nützlich für die Durchführung 6 war. Der Inhalt der Arbeit wurde mithilfe von Herrn Brekenfeld nochmals begutachtet, um mögliche große Fehler zu vermeiden. Daraufhin wurde das Tool zur Kontrolle nochmals auf seine Funktionalität geprüft. Zudem wurde mehrmals die schriftliche Ausarbeitung überarbeitet und die letzten Formalitäten eingebunden.

6.3 Evaluierung des Tools

In diesem Kapitel wird die Daseinsberechtigung des Tools ermittelt. Dabei werden die manuelle und die automatisierte Suche gegenübergestellt Die manuelle Suche beschreibt den Vorgang des eigenhändigen Eingebens und Durchsuchen der Treffer, während die automatisierte Suche die Verwendung des Tools beschreibt. Die Treffer der Suchanfrage unterscheiden nicht zwischen manueller und automatisierter Eingabe und bilden somit eine identische Ergebnismenge. Dass eine manuelle Suche grundsätzlich den gleichen Effektivitätsgrad wie eine automatisierte Suche liefern kann, ist auch erst als Forschungsprodukt bei der Entwicklung des Tools ersichtlich geworden. Das Verwenden von Schlüsselwörtern als Hauptindikator für relevante Issues hat durch ihre in 3.4.1 und 3.4.5 beschriebene Auswertung ihre potente Effektivität zugesprochen bekommen. Dabei wurden nach eigener Einschätzung nach den Ergebnissen händisch ein Rang je nach Effektivitätsgrad zugewiesen und anschließend mit Herrn Brekenfelds Beurteilung abgeglichen und bestätigt. Das ausschlaggebendste Argument für das Tool sind die Metriken, wie Kommentaranzahl, die anfrageungebunden sind. Dadurch erlauben sie dem Tool bereits vor der eigentlichen Ausgabe die Ergebnisse zu filtern. Somit muss nicht für jedes individuelle Issue manuell entschieden werden, ob dieses den gewollten Metriken entspricht. Dies geschieht auf eine Weise, die es der zeitaufwendigen und eigenhändigen Untersuchung unmöglich macht mit der automatisierten mitzuhalten. Zum anderen bildet es durch das Caching und der manuellen Bewertung eine Fortschrittsübersicht, die einem bei Bedarf nur ungelesene Issues ausgibt. Dieser Fortschritt wäre sonst aufgrund von Überprüfung und sich ändernder Reihenfolge der Treffer nur sehr mühsam zu erreichen. Zusätzlich werden bei der Ausgabe des Tools die Ergebnisse so aufbereitet ausgegeben, dass die bereitgestellten Informationen dem Anwender wertvollen Kontext und Hinweise zum Interessengrad des Issues liefern können. Diese Informationssammlung manuell zu erstellen, würde kostbare Zeit kosten, die nicht in das Lesen des eigentlichen Issues investiert werden würde. Zusammenfassend profitiert das Tool also durch Benutzerfreundlichkeit und enorme Effizienz in Bezug auf Zeit als Ressource.

7 Fazit und Ausblick

7.1 Fazit

Im Rahmen dieser Arbeit wurde ein Tool entwickelt, das für die qualitative Forschung im Bereich „Technical Debt“ vorgesehen ist. Der Fokus liegt auf der Bereitstellung von Diskussionen aus Issues von Github. Nach einem Wechsel des Lösungsansatzes nach der Hälfte der Arbeitszeit konnte durch Wiederverwendung von Wissen und Quelltext dennoch ein fertiges Endprodukt umgesetzt werden. Dabei sind mehrere Lösungsansätze untersucht, abgewägt und entwickelt worden. Letztlich wurde sich für die „Google Search JSON API“ in Kombination mit der „Google Custom Search Engine“ entschieden. Nach einigen Effektivitätstests zeigte sich eine Filterung der Diskussionen über Schlüsselwörter als hochwertigste Metrik. Das Tool ist durch die Einschätzung des Betreuers, welcher hauptsächlich das Tool verwenden wird, erfolgreich umgesetzt worden.

7.2 Ausblick

Das Tool setzt die erwünschten Kernfunktionalitäten um, dennoch bietet es Erweiterungsmöglichkeiten. Einige, wie weitere Metriken zur zusätzlichen Einschränkung, werden vermutlich erst bei der praktischen Anwendung des Tools ersichtlich werden. Andere hingegen sind bereits, wie in 4.2.2 erwähnt, bekannt. Der Zugriff auf die Datenbank stellt dabei einen erweiterbaren Bereich dar, weil dieser nicht vollständig von den implementierten Methoden des interaktiven Modus abgedeckt wird. Das Tool soll langfristig seine Effektivität mit dem Anwenden steigern. Dies soll erreicht werden, indem durch empirische Durchläufe eine immer effektivere Permutation von Anfragen und Metriken gebildet wird. Außerdem könnte das Tool als Fundament für die Entwicklung weiterer Tools im qualitativen Forschungsbereich dienen.

Literatur

[1] W. Cunningham, „The wycash portfolio management system, experience report," *Proceedings on Object-oriented programming systems, languages, and applications (OOPSLA'92)*, 1992.

[2] Y. Gueheneuc, „Ptidej: A Flexible Reverse Engineering Tool Suite," in *2007 IEEE International Conference on Software Maintenance*, 2007, S. 529–530.

[3] S. Wong, Y. Cai, M. Kim und M. Dalton, „Detecting software modularity violations," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, S. 411–420.

[4] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler und J. Penix, „Using static analysis to find bugs," *IEEE software*, Jg. 25, Nr. 5, S. 22–29, 2008.

[5] J.-L. Letouzey, „The SQALE method for evaluating technical debt," in *2012 Third International Workshop on Managing Technical Debt (MTD)*, IEEE, 2012, S. 31–36.

[6] M. J. Rochkind, „The source code control system," *IEEE transactions on Software Engineering*, Nr. 4, S. 364–370, 1975.

[7] A. E. Hassan, R. C. Holt und A. Mockus, „MSR 2004: International workshop on mining software repositories," in *Proceedings of the 26th International Conference on Software Engineering*, 2004, S. 770–771.

[8] *MSR 2020*. Adresse: https://2020.msrconf.org/ (besucht am 18. 10. 2020).

[9] G. Gousios, „The GHTorrent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, Ser. MSR '13, San Francisco, CA, USA: IEEE Press, 2013, S. 233–236, ISBN: 978-1-4673-2936-1. Adresse: http://dl.acm.org/citation.cfm?id=2487085.2487132.

[10] D. Spadini, M. Aniche und A. Bacchelli, „PyDriller: Python framework for mining software repositories," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, New York, New York, USA: ACM Press, 2018, S. 908–911, ISBN: 9781450355735. DOI: 10.1145/3236024.3264598. Adresse: http://dl.acm.org/citation.cfm?doid=3236024.3264598.

[11] C. G. Campos, *Cvsanaly*, 2009.

[12] R. Di Cosmo, „How to use Software Heritage for archiving and referencing your source code: guidelines and walkthrough,“ *arXiv preprint arXiv:1909.10760*, 2019.

[13] L. Neil, S. Mittal und A. Joshi, „Mining threat intelligence about open-source projects and libraries from code repository issues and bug reports,“ in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, IEEE, 2018, S. 7–12.

[14] E. d. S. Maldonado, E. Shihab und N. Tsantalis, „Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt,“ *IEEE Transactions on Software Engineering*, Jg. 43, Nr. 11, S. 1044–1062, 2017.

[15] S. Mhatre, *The untold story of Github*, en, Okt. 2016. Adresse: https://medium.com/@smhatre59/the-untold-story-of-github-132840f72f56 (besucht am 22. 09. 2020).

[16] *Build software better, together*, en. Adresse: https://github.com (besucht am 22. 09. 2020).

[17] *Build software better, together*, en. Adresse: https://github.com/search (besucht am 22. 09. 2020).

[18] *github.com Competitive Analysis, Marketing Mix and Traffic - Alexa*. Adresse: https://www.alexa.com/siteinfo/github.com (besucht am 22. 09. 2020).

[19] *Custom Search JSON API | Programmable Search Engine*, en. Adresse: https://developers.google.com/custom-search/v1/overview (besucht am 25. 09. 2020).

[20] *Custom Search JSON API: Introduction | Programmable Search Engine*, en. Adresse: https://developers.google.com/custom-search/v1/introduction (besucht am 25. 09. 2020).

[21] *SQLite is a Self Contained System*. Adresse: https://www.sqlite.org/selfcontained.html (besucht am 25. 09. 2020).

[22] *Querying MongoDB programmatically*. Adresse: https://ghtorrent.org/raw.html (besucht am 18. 10. 2020).

[23] *Downloads*. Adresse: https://ghtorrent.org/downloads.html (besucht am 20. 10. 2020).

[24] *The Python Database API (DBAPI) 2.0 - Python in a Nutshell, 2nd Edition [Book]*, en. Adresse: https://www.oreilly.com/library/view/python-in-a/0596100469/ch11s04.html (besucht am 06. 10. 2020).

[25] A. Wieland und C. M. Wallenburg, „Dealing with supply chain risks," *International Journal of Physical Distribution & Logistics Management*, 2012.

[26] *Implementation Limits For SQLite*. Adresse: https://www.sqlite.org/limits.html (besucht am 13. 10. 2020).