

Institut für Softwaretechnologie

Universität Stuttgart  
Universitätsstraße 38  
D-70569 Stuttgart

Bachelorarbeit

# **Automatische Durchführung von Refactorings durch Kommentare in GitHub Pull-Requests**

Stefan Basaric

**Studiengang:** Softwaretechnik  
**Prüfer:** Prof. Dr. Stefan Wagner  
**Betreuer:** Marvin Wyrich, M.Sc.

**Beginn am:** 28. November 2018  
**Beendet am:** 28. Mai 2019



## **Kurzfassung**

Um das Refactoring von JAVA-Projekten automatisieren zu können, wurde der sogenannte Refactoring-Bot entwickelt. Der Refactoring-Bot ist ein JAVA-Projekt, welches in der Lage ist Bots zu konfigurieren, welche sich Daten einer statischen Codeanalyse eines GitHub-Projekts holen, um anschließend die in der Analyse gefundenen Code-Smells automatisiert zu beheben. Der angepasste Code wird zum Schluss den Projektbesitzern in Form eines Pull-Requests vorgeschlagen.

In dieser Arbeit wurde der Refactoring-Bot um die Implementierung eines kommentargesteuerten Refactorings erweitert. Dieses soll den GitHub-Projektverantwortlichen ermöglichen, den Refactoring-Bot mittels Kommentaren anzuweisen, seine Pull-Requests nachträglich zu bearbeiten. Zusätzlich wurde eine qualitative Studie durchgeführt, bei welcher Testpersonen die neue Funktionalität auf einem Beispielprojekt testen konnten, welches auf GitHub gehostet ist. Das Ergebnis der Studie zeigt, dass das kommentargesteuerte Refactoring, trotz einiger Mängel, überwiegend positiv bei den Testpersonen angekommen ist.

## **Abstract**

In order to automate the refactoring process of JAVA projects, a Refactoring-Bot has been developed. The Refactoring-Bot is a JAVA project, that is able to configure bots, that fetch data from a static code analysis of a GitHub project. The bot then automatically fixes the code smells that were returned by the analysis service. Finally, the bot recommends the refactored code to project owners in form of a pull request.

In this work, the Refactoring-Bot was enhanced with the implementation of a comment-driven refactoring. The idea is to allow project owners of GitHub projects to use the power of pull request comments to instruct the Refactoring-Bot to make further changes inside his pull request. In addition, a qualitative study was conducted, in which subjects were able to test the new functionality with a sample project, that was hosted on GitHub. The result of the study shows that the comment-driven refactoring, despite some shortcomings, has gathered mostly positive reception among the subjects.



# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>11</b>
1.1. Motivation . . . . .	11
1.2. Aufgabenstellung und Ziel . . . . .	12
1.3. Aufbau der Arbeit . . . . .	12
<b>2. Grundlagen</b>	<b>13</b>
2.1. Refactoring . . . . .	13
2.2. Pull-basierte Softwareentwicklung . . . . .	13
2.3. Software-Bots . . . . .	15
<b>3. Implementierung</b>	<b>21</b>
3.1. Grundlagen . . . . .	21
3.2. Tools und Services . . . . .	22
3.3. Relevante JAVA-Objekte . . . . .	29
3.4. Plattformunabhängige Datensammlung . . . . .	31
3.5. Kommentarverarbeitung . . . . .	32
3.6. Durchführung der automatisierten Refactorings . . . . .	34
3.7. Implementierung neuer Refactorings . . . . .	35
3.8. Anleitung und Feedback . . . . .	35
<b>4. Evaluation</b>	<b>39</b>
4.1. Ziele . . . . .	39
4.2. Vorbereitung . . . . .	39
4.3. Durchführung . . . . .	40
<b>5. Ergebnisse</b>	<b>43</b>
5.1. Testpersonen . . . . .	43
5.2. Pull-Request-Aufgaben . . . . .	45
5.3. Evaluationsbogen . . . . .	47
<b>6. Diskussion</b>	<b>51</b>
6.1. Ergebnisse . . . . .	51
6.2. Limitationen . . . . .	54
<b>7. Fazit</b>	<b>57</b>
<b>Literaturverzeichnis</b>	<b>59</b>
<b>A. Einverständniserklärung</b>	<b>61</b>
<b>B. Aufgabenblatt</b>	<b>63</b>



# Abbildungsverzeichnis

2.1.	Beispiele für ein Pull-Request . . . . .	14
2.2.	Beispiel eines Pull-Request-Reviews . . . . .	15
3.1.	Funktionsweise des Refactoring-Bots . . . . .	21
3.2.	Beispiel für die Funktionsweise eines Lexers . . . . .	24
3.3.	Beispiel für einen Syntaxbaum eines Parsers . . . . .	24
3.4.	Beispiel eines Traits . . . . .	26
3.5.	Beispiel: Erkennung durch freien Text . . . . .	27
3.6.	Fehlerhafte Erkennung eines Textes . . . . .	27
3.7.	Korrigierte Bedeutung eines Textes . . . . .	28
3.8.	Inbox einer Wit-Anwendung . . . . .	28
3.9.	Entitäten der Refactoring-Sprache . . . . .	29
3.10.	Plattformunabhängiges Sammeln der Pull-Request-Kommentare . . . . .	32
3.11.	Übersetzung der Kommentare . . . . .	33
3.12.	Workflow des automatisierten Refactorings . . . . .	34
3.13.	Beispiel einer Nutzungsanleitung . . . . .	36
3.14.	Beispiele von Fehlernachrichten . . . . .	37
5.1.	Erfahrung mit JAVA . . . . .	43
5.2.	Regelmäßige Nutzung von Filehostern . . . . .	44
5.3.	Genutzte Filehoster . . . . .	44
5.4.	Wichtigkeit von Software-Refactoring . . . . .	45
5.5.	Ergebnisse der Pull-Request-Bearbeitung . . . . .	45
5.6.	Fehlergründe für nicht durchgeführte Refactorings . . . . .	46
5.7.	Kommentar-Anzahl in bearbeiteten Pull-Requests . . . . .	46
5.8.	Evaluationsergebnisse der Bedienung . . . . .	47
5.9.	Evaluationsergebnisse der Sprachverarbeitung . . . . .	48
5.10.	Evaluationsergebnisse des Bot-Feedbacks . . . . .	48
5.11.	Gesamteindruck vom kommentargesteuerten Refactoring . . . . .	49
6.1.	Nicht verstandener Kommentar . . . . .	53
6.2.	Sehr detaillierte Anweisung . . . . .	53
6.3.	Beispiel einer schlechten Spracherkennung . . . . .	54
6.4.	Beispiel einer möglichen Manipulation . . . . .	56





## Verzeichnis der Listings

3.1. Beispiel für eine ANTLR-Grammatik . . . . .	23
3.2. Refactoring-Beispiele mittels ANTLR . . . . .	25



# 1. Einleitung

## 1.1. Motivation

Die Idee, dass eine Maschine ein dem Menschen gleichwertiges Denkvermögen hat, wurde schon in den 50er Jahren von Alan Turing in seinem Paper [Tur50] vorgeschlagen. Bei seinem sogenannten Turing-Test geht es darum, einem Menschen vorzutäuschen, dass er mit einem anderen Menschen statt einer Maschine, einem sogenannten Software-Agenten oder auch Software-Bot, kommuniziert.

Seitdem wird das Ziel verfolgt, solche intelligenten Software-Agenten zu entwickeln. Durch die stetige Forschung und Entwicklung im Bereich der künstlichen Intelligenz und der Computerlinguistik gibt es heutzutage eine große Menge an verschiedensten Software Bots [WSS+18], welche ein großes Spektrum an Aufgaben in verschiedensten Bereichen durchführen. Im Vergleich zur ursprünglichen Idee des Turing-Tests, ist das Ziel des Bots heutzutage nicht dem Menschen vorzutäuschen keine Software zu sein, sondern ihn beim Arbeiten zu unterstützen oder gängige, sich wiederholende Aufgaben automatisiert durchzuführen [LSZ18].

Ein Ort, an dem sich viele solcher Software-Agenten befinden [WSS+18], ist der Onlinedienst GitHub<sup>1</sup>. GitHub ist einer der größten Filehoster mit vielen Millionen von Nutzern und öffentlichen Repositories. Eines der vielen Repositories enthält den sogenannten Refactoring-Bot<sup>2</sup>, welcher am ISTE der Universität Stuttgart entwickelt wurde [WB19]. Der Refactoring-Bot ist ein JAVA-Projekt, welches seinen Nutzern ermöglicht einen Bot anzuweisen, an anderen Projekten auf GitHub mitzuwirken. Dafür sammelt der Bot zunächst alle Befunde einer statischen Codeanalyse für ein bestimmtes Projekt. Um die Befunde zu korrigieren, passt der Bot den fehlerhaften Code an, um zum Schluss einen Pull-Request auf das Projekt erstellen zu können.

Ist der Pull-Request erstellt, so ist die Wahrscheinlichkeit hoch, dass er früher oder später von einem Projektverantwortlichen analysiert wird. Findet der Projektverantwortliche bestimmte Mängel im vom Bot angepassten Code, so wird er entweder den Pull-Request schließen, oder bestimmte Änderungen in einem Pull-Request-Review fordern. Das beurteilen von Code in einem Review erfolgt auf GitHub in Form von Zeilenkommentaren [Git19d].

Da die aktuelle Implementierung des Refactoring-Bots keine Sprachverarbeitung unterstützt, wird der Bot niemals auf die Kommentare des Projektverantwortlichen eingehen, und damit auch den Änderungsforderungen nicht nachkommen. Das stellt den Projektverantwortlichen vor die Wahl, entweder den Pull-Request des Bots zu schließen, oder die Änderungen selbst vorzunehmen.

---

<sup>1</sup><https://github.com>

<sup>2</sup><https://github.com/Refactoring-Bot/Refactoring-Bot>

Um dem Projektverantwortlichen die Arbeit zu erleichtern ist eine Funktionalität erforderlich, welche ihm ermöglicht, dem Bot Anweisungen zu geben, um bestimmte Änderungen am Pull-Request vorzunehmen.

### **1.2. Aufgabenstellung und Ziel**

Im Rahmen dieser Bachelorarbeit soll der vorgestellte Refactoring-Bot um die Funktionalität erweitert werden, auf Kommentare in den von ihm geöffneten Pull-Requests reagieren zu können. Dabei soll der Refactoring-Bot alle an ihn gerichteten Kommentare verstehen, welche Codeänderungen fordern. Anhand des Kommentars soll er das passende Refactoring an passender Stelle durchführen und den Pull-Request mit den Änderungen aktualisieren. Auf diese Weise wäre ein Projektverantwortlicher in der Lage, den Bot anzuweisen gewünschte Änderungen am Code vorzunehmen, ohne den Code des Pull-Requests selber manuell bearbeiten zu müssen. Zum Schluss soll in einer Studie überprüft werden, wie gut das implementierte Interaktionsmodell des kommentargesteuerten Refactoring-Bots bei ausgewählten Testpersonen, die aus einem passenden Fachgebiet kommen und nötige Kenntnisse haben, ankommt.

### **1.3. Aufbau der Arbeit**

Im 2. Kapitel werden die für diese Arbeit relevanten Begriffe definiert und wichtige Details zu Chatbots und ihrer Implementierung aus verwandten Arbeiten beschrieben. Das darauffolgende Kapitel 3 beschreibt die Implementierung einer Lösung des kommentargesteuerten Refactorings. Im 4. Kapitel wird eine Studie beschrieben, welche im Rahmen dieser Arbeit für die Evaluation der Implementierung des kommentargesteuerten Refactorings durchgeführt wurde. Anschließend werden in Kapitel 5 die Ergebnisse dieser Studie vorgestellt und in Kapitel 6 diskutiert. Zum Schluss werden in Kapitel 7 die wichtigsten Punkte der Arbeit zusammengefasst und ein Ausblick auf zukünftige Arbeiten gegeben.

## 2. Grundlagen

In diesem Kapitel werden relevante Begriffe beschrieben, welche für diese Arbeit eine wichtige Rolle spielen. Darüber hinaus werden auch wichtige Informationen zu Chatbots und deren Implementierung dargestellt.

### 2.1. Refactoring

Bei einem Refactoring ist es das Ziel die interne Struktur einer Software anzupassen ohne dabei ihre Funktionalität zu verändern. Die durch ein Refactoring verbesserte Codestruktur sorgt dabei für eine Steigerung ihrer Lesbarkeit und Verständlichkeit, was wiederum eine effizientere Fehlersuche, Wartung und Weiterentwicklung der Software ermöglicht [Gmb19].

### 2.2. Pull-basierte Softwareentwicklung

In diesem Unterkapitel werden alle wichtigen Begriffe aus der pull-basierten Softwareentwicklung erläutert.

#### 2.2.1. Repository

Ein Repository ist ein Verzeichnis, welches alle Dateien eines Projekts mitsamt ihrer Revisionsgeschichte enthält [Git19c].

#### 2.2.2. Fork

Gängige Filehoster wie GitHub ermöglichen ihren Nutzern auf einem sehr einfachen Weg an anderen Projekten mitzuwirken. Dies geschieht über sogenannte Repository-Forks. Ein Fork ist eine Kopie eines Repositories. Im Falle von öffentlichen Projekten können GitHub Nutzer von diesen Projekten ohne Einschränkungen Forks generieren. Dabei verwaltet GitHub den Quellcode und die Commits eines jeden Forks separat. Dadurch können alle Nutzer auf ihren Forks bestimmte Module implementieren und anpassen, ohne jede Codeänderung beim originalen Repository melden zu müssen. Auf diese Weise können auf den Forks vollständige Funktionen implementiert werden, bevor sie den Projektverantwortlichen des originalen Repository in einem Pull-Request vorgeschlagen werden [RR14].

## 2. Grundlagen

### 2.2.3. Pull-Request

Ein Pull-Request ermöglicht seinem Ersteller anderen Entwicklern alle getätigten Änderungen seines Git-Branche seit der letzten Synchronisierung mit dem Ziel-Branch mitzuteilen. Im Pull-Request können die Änderungen mit anderen Mitwirkenden überprüft, besprochen und angepasst werden [Git19b]. Im Schaubild 2.1 sind zwei Ansichten eines Pull-Requests aufgeführt. Die erste Ansicht zeigt die Hauptseite eines Pull-Requests, welche grundlegende Informationen zu den getätigten Änderungen enthält. In der zweiten Ansicht sieht man die Änderungen aller Dateien, welche in den Commits des Pull-Requests durchgeführt wurden.

## Extracted method from 'areCoprime' #3

The screenshot shows a GitHub Pull Request interface. At the top, it says 'LHommeDeBot wants to merge 1 commit into LaPersonneDeTest:master from LHommeDeBot:extractMethod'. Below this, there are statistics: Conversation (0), Commits (1), Checks (0), and Files changed (1). The main content area shows a comment from 'LHommeDeBot' 10 days ago, edited by 'LaPersonneDeTest', with the text 'No description provided.' Below the comment is a commit titled 'Extracted method from 'areCoprime'' with the hash 'efdf03b'. A message below the commit says 'Add more commits by pushing to the extractMethod branch on LHommeDeBot/Studie10.' A green box contains a 'Merge pull request' button and a message: 'Continuous integration has not been set up. Several apps are available to automatically catch bugs and enforce style. This branch has no conflicts with the base branch. Merging can be performed automatically. You can also open this in GitHub Desktop or view command line instructions.'

(a) Hauptseite eines Pull-Requests

The screenshot shows a diff view of a file named 'src/services/Calculator.java'. The code is shown with line numbers 12 to 18. Line 15 shows a change from a red line to a green line. The red line is 'public final static double PI = 3.14159265359;' and the green line is 'public static final double PI = 3.14159265359;'. Line 16 shows 'public final static double e = 2.718281828459045;'. Line 18 shows '/\*'. The diff shows a '-' sign for the red line and a '+' sign for the green line.

(b) Codeänderungen eines Pull-Requests

Abbildung 2.1.: Beispiele für ein Pull-Request

### 2.2.4. Pull-Request-Review

Pull-Request-Reviews ermöglichen allen Mitwirkenden über Codeänderungen eines Pull-Requests zu diskutieren. Darüber hinaus ermöglichen sie allen Mitwirkenden und Projektverantwortlichen Änderungen am Pull-Request mithilfe von Zeilenkommentaren zu fordern. Dies ermöglicht dem Author des Pull-Requests diesen so anzupassen, sodass dieser alle Qualitätsstandards des Projekts erfüllt, bevor seine Änderungen zum Projekt hinzugefügt werden [Git19d] [Git19a]. Im Schaubild 2.2 wird ein Beispiel eines Pull-Request-Reviews gezeigt.

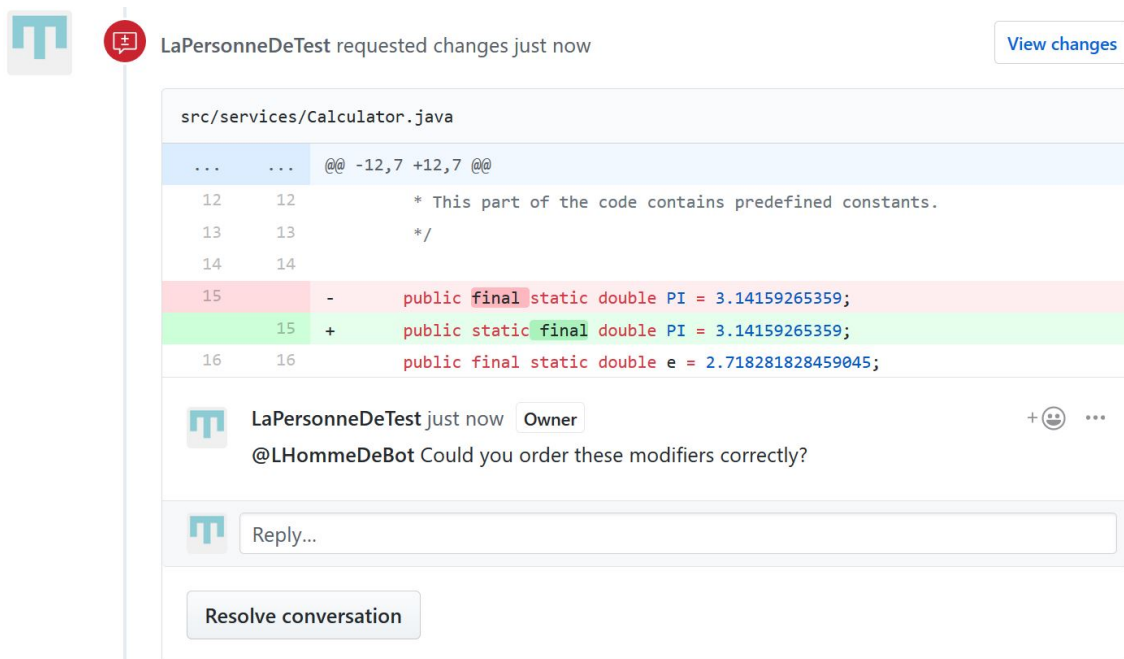


Abbildung 2.2.: Beispiel eines Pull-Request-Reviews

## 2.3. Software-Bots

Dieses Unterkapitel enthält alle wichtigen Informationen zu Software-Bots.

### 2.3.1. Grundlagen

Ein Bot ist ein System, welches selbstständig agieren und bestimmte Aufgaben durchführen kann. Die Aufgabe der Bots ist es zu versuchen den Menschen bei bestimmten Tätigkeiten zu unterstützen. So können sie die Kommunikation und Kollaboration zwischen Menschen verbessern, Informationen sammeln und präsentieren, oder wiederkehrende Aufgaben automatisiert lösen [LSZ18].

Beispielsweise können manche Bots veröffentlichten Code analysieren und bewerten, indem sie Entwickler auf mögliche Bugs und Code-Smells<sup>1</sup> aufmerksam machen. Andere können Pull-Requests analysieren<sup>2</sup> oder Fehler bei der kontinuierlichen Integration<sup>3</sup> melden.

Eine Unterkategorie von Bots bilden die sogenannten Chatbots. Dabei handelt es sich um Software-Agenten, welche in der Lage sind mit Menschen mittels Stimme oder Text zu kommunizieren [Pv18]. So können Chatbots beispielsweise neue Mitwirker eines Projekts mit einer Nachricht begrüßen<sup>4</sup> oder diese auffordern ein Contributor License Agreement, kurz CLA, zu unterschreiben<sup>5</sup>. Weitere bekannte Beispiele für Chatbots sind intelligente Assistenten wie Siri oder Cortana.

### 2.3.2. Interaktion

Für die Implementierung des Interaktionsmodells eines Chatbots hat der Entwickler mehrere Möglichkeiten bei der Auswahl der Art der Initiierung, der Kommunikationsrichtung, der Sprache und des Kommunikationskanals.

Bei der Initiierung der Kommunikation gibt es dabei zwei Vorgehensweisen. Bei der ersten Vorgehensweise initiiert der Bot die Konversation basierend auf den Daten eines Systems oder eines Kontexts. Ein Beispiel hierfür ist die aktuelle Implementierung des Refactoring-Bots, in welcher der Bot automatisch bei neuen Sonarcloud-Issues neue Pull-Requests erstellt um mit den Projektverantwortlichen eines Projekts zu kommunizieren. Bei der anderen Vorgehensweise initiiert der Mensch die Kommunikation mit einem Bot. Dabei kann der Mensch mittels Stimme oder Text einem Bot eine Anweisung geben, welche vom Bot bearbeitet werden soll [LSZ18]. Nach einer erfolgreichen Initiierung kann die Kommunikation zwischen Bot und Mensch entweder unidirektional oder bidirektional verlaufen [Pv18].

Für die Auswahl der Kommunikationssprache hat der Entwickler die Wahl zwischen einer natürlichen oder einer domänenspezifischen Sprache [LSZ18]. Bei der domänenspezifischen Sprache verwendet man für die Interaktion mit dem Bot spezielle Befehle, welche vom Bot verstanden werden können. Im Vergleich zur strikten, domänenspezifischen Sprache ist es auch möglich mittels einer natürlichen Sprache sehr frei und realitätsnah mit einem Bot zu kommunizieren. Diese freien Konversationen, wie man sie normalerweise nur unter Menschen beobachten kann, müssen zunächst verarbeitet werden. Im Bereich der Computerlinguistik gibt es dafür schon einige Tools, Frameworks und Services, welche diese Verarbeitung von Text und Stimme ermöglichen.

Die Verarbeitung von Sprache kann über einen Text erfolgen, welcher üblicherweise über eine Messaging-Plattform wie Facebook, GitHub oder Slack veröffentlicht wird. Alternativ ist es auch möglich, die Kommunikation mittels Stimmerkennung durchzuführen. Beispiele für eine akustische Kommunikation sind virtuelle Assistenten wie Cortana oder Siri. Zuletzt gibt es auch Bots die diese beiden Kommunikationskanäle kombinieren und die Kommunikation mittels Text und Stimme gleichzeitig ermöglichen [Pv18].

---

<sup>1</sup><https://github.com/houndci/hound>

<sup>2</sup><https://github.com/twbs/rorschach>

<sup>3</sup><https://github.com/travis-ci/travis-ci>

<sup>4</sup><https://github.com/apps/the-welcome-bot>

<sup>5</sup><https://github.com/apps/cla-bot>



### 2.3.3. Tools für Computerlinguistik

Durch die immer weiter voranschreitende Forschung im Bereich der künstlichen Intelligenz und Computerlinguistik steigt auch die Beliebtheit der immer natürlicher agierenden Chatbots. Heutzutage gibt es ein breites Spektrum an Frameworks, Tools und Services, welche dem Entwickler wichtige Funktionen und Werkzeuge liefern, damit Sie schnell und einfach Chatbots implementieren und verwenden können. Diese Services oder Tools können sich stark voneinander in ihrer Funktionsvielfalt und Spezialisierung unterscheiden [LSZ18]. So ermöglichen einige Tools den Entwicklern ihre erstellten Bots auf beliebigen Plattformen oder in beliebigen Projekten zu integrieren, während andere den erstellten Bot an bestimmte Plattformen binden. Manche Tools bieten nur eine Verarbeitung der natürlichen Sprache an, während andere dem Entwickler ermöglichen einen kompletten Bot zu implementieren und diesen auf einer Plattform zu deployen. Die Entwicklung von Bots kann mit manchen Tools ohne jegliche Programmierkenntnisse durchgeführt werden, während andere Tools auf Dokumentation und vorgefertigte Codetemplates setzen und damit eine bestimmte Menge an Programmiererfahrung bei ihren Entwicklern voraussetzen. Um Entwickler beim Einstieg in die Entwicklung der Chatbots zu unterstützen, bieten manche Tools auch digitale Gemeinschaften an, wo sich Neulinge mit den Entwicklern des Tools oder erfahrenen Nutzern in Verbindung setzen können. In diesen Gemeinschaften können Probleme diskutiert, oder Verbesserungen vorgeschlagen werden.

Ein beliebter, von Facebook angebotener Webservice für Computerlinguistik ist Wit.ai<sup>6</sup>. Wit.ai ermöglicht dem Entwickler sogenannte Wit-Projekte über ein Web-Interface zu erstellen und zu trainieren. Diese Projekte sind in der Lage die natürliche Sprache zu verarbeiten. Der komplette Entwicklungsprozess der Sprachverarbeitung fordert dem Entwickler keine Programmierkenntnisse ab und eignet sich damit auch für Softwareentwickler ohne Erfahrungen auf dem Gebiet der Computerlinguistik und künstlichen Intelligenz. Mit Wit.ai können Entwickler ihre Bots zu kompetenten Chatbots umwandeln, welche in der Lage sind verschiedenste natürliche Sprachen mittels Text oder Stimme zu verarbeiten.

Alternativ können sich Entwickler von Chatbots auch an dem von Google angebotenen Webservice Dialogflow<sup>7</sup> bedienen. Genau wie Wit.ai bietet Dialogflow dem Entwickler die Möglichkeit, schnell und einfach eine natürliche Sprachverarbeitung mittels eines Web-Interfaces umzusetzen. Im Vergleich zu Wit.ai besitzt Dialogflow jedoch ein Preismodell. So sind einige Funktionalitäten bei einer kostenfreien Nutzung von Dialogflow, wie zum Beispiel die Verarbeitung von natürlicher Sprache mittels Stimme, eingeschränkt<sup>8</sup>.

Luis.ai<sup>9</sup> ist ein weiterer, von Microsoft angebotener, Webservice für Computerlinguistik. Wie Wit.ai oder Dialogflow ermöglicht Luis.ai die Verarbeitung natürlicher Sprache mittels Text oder Stimme. Analog zu den oben genannten Webservices verlangt auch Luis.ai den Entwicklern keine Kenntnisse im Bereich der Computerlinguistik ab, da auch hier das Training der Chatbots über ein simples, grafisches Web-Interface erfolgt. Ähnlich wie Dialogflow ist Luis.ai für die nicht zahlenden Nutzer

---

<sup>6</sup><https://wit.ai/>

<sup>7</sup><https://dialogflow.com/>

<sup>8</sup><https://dialogflow.com/pricing>

<sup>9</sup><https://www.luis.ai/home>

## 2. Grundlagen

---

in seiner Funktionalität eingeschränkt. So ist zum einen die Verarbeitung der natürlichen Sprache mittels Stimme nur für zahlende Kunden verfügbar und zum anderen ist die Anzahl der Anfragen für eine textuelle Sprachverarbeitung eingeschränkt<sup>10</sup>.

Die oben genannten Webservices sind nur für die Verarbeitung der natürlichen Sprache zuständig. Sie erstellen also keine funktionierenden Bots, sondern ermöglichen schon vorhandenen Bots von einer natürlichen Sprachverarbeitung zu profitieren. Einfach gesagt, wandeln diese Webservices einen einfachen Bot in einen Chatbot um. Im Vergleich zu den oben genannten Services gibt es auch Services die dem Entwickler ermöglichen einen funktionstüchtigen Chatbot zu implementieren. Beispiele für solche Services sind Chatfuel<sup>11</sup> oder ManyChat<sup>12</sup>. Diese Services sind jedoch in der Nutzbarkeit stark eingeschränkt, denn die erstellten Chatbots können nur in Verbindung mit dem Facebook Messenger<sup>13</sup> verwendet werden. Dazu besitzen die beiden Services keine eigene natürliche Sprachverarbeitung wie man sie mit Dialogflow oder Wit.ai erhält, sodass die erstellten Chatbots keine komplexeren Aussagen der Nutzer verstehen können. Die Möglichkeit besteht aber auch hier, diese Services mit beispielsweise Dialogflow zu kombinieren<sup>14</sup>, sodass auch diese Chatbots von der natürlichen Sprachverarbeitung profitieren können.

### 2.3.4. Softwarebots in der Praxis

In einer Studie [WSS+18] wurden quelloffene GitHub-Projekte auf Software-Bots untersucht. Dabei wurde überprüft, wie viele der öffentlichen und bekannten GitHub-Projekte Bots verwenden und welche Funktionen jene Bots im jeweiligen Projekt haben. Darüber hinaus wurde eine Befragung der Entwickler, welche an diesen Projekten gearbeitet haben, durchgeführt. Die Befragung hatte den Zweck, alle Herausforderungen und gewünschte Features im Bezug auf Software Bots, welche in Pull-Requests aktiv sind, zu finden.

Die Ergebnisse der Studie zeigen, dass nicht alle Entwickler mit den Bots zufrieden sind. Die Probleme, die von den Entwicklern am häufigsten genannt wurden beziehen sich entweder auf die Intelligenz der Bots, oder auf deren Feedback. Im Bezug auf die mangelnde Intelligenz beschrieben manche Entwickler, dass Bots eher für Probleme sorgen, anstatt diese zu lösen. Andere Entwickler sagen, dass Bots von vergangenen Interaktionen nicht lernen und die gleichen Fehler wiederholt tätigen. Darüber hinaus sind die Bots laut einigen Entwicklern nicht in der Lage gutes Feedback zu geben. Dabei kann das vom Bot kommende Feedback entweder zu unfreundlich, irreführend, oder ungenau sein. So sagen einige Entwickler, dass manche Bots im Falle von Problemen ein zu unklares Feedback geben, sodass die Entwickler nicht verstehen konnten, wie dieses Problem gelöst werden könnte. Ergänzend zum Feedback klagen einige Entwickler über Probleme bei der Kommunikation mit einigen Bots, da diese dem Entwickler nicht klar machen, wie man mit ihnen interagiert, sodass die Interaktion zwischen Bot und Entwickler gestört ist.

---

<sup>10</sup><https://azure.microsoft.com/en-us/pricing/details/cognitive-services/language-understanding-intelligent-services/>

<sup>11</sup><https://chatfuel.com/>

<sup>12</sup><https://manychat.com/>

<sup>13</sup><https://www.messenger.com/>

<sup>14</sup><https://chatbotsmagazine.com/integrate-manychat-with-dialogflow-with-context-sharing-nlp-with-manychat-7e9d46648cb9>

Die größte Forderung der Entwickler sind schlauere Bots, welche aus vergangenen Interaktionen lernen und den Entwickler besser unterstützen. Ein weiterer Wunsch der Entwickler sind Bots mit mehreren und besseren Interaktionsmöglichkeiten. Neben der Verbesserung der Interaktionsmöglichkeiten wünschen sich einige Entwickler auch einfachere Wege diese zu erlernen, sodass sich vor allem neue Nutzer, schnell und einfach mit den Interaktionsmöglichkeiten des Bots vertraut machen können. Des Weiteren wünschen sich Entwickler bessere Benachrichtigungen, welche den Nutzern klar machen, dass beispielsweise Probleme aufgetaucht, oder Aufgaben abgeschlossen sind.



## 3. Implementierung

In diesem Kapitel wird die Implementierung eines kommentargesteuerten Refactorings vorgestellt. Dabei wird auch die Funktionsweise verschiedener Tools und Services erläutert, welche bei der Implementierung eine wichtige Rolle spielen.

### 3.1. Grundlagen

Der Refactoring-Bot ist ein JAVA Projekt, welches mit dem Spring Boot<sup>1</sup> Framework geschrieben wurde. Für das Management der benötigten Abhängigkeiten verwendet das Projekt das Build-Management-Tool Apache Maven<sup>2</sup>. Die Persistierung von Daten erfolgt in einer MySQL-Datenbank<sup>3</sup>.

Mit dem Refactoring-Bot ist ein Nutzer in der Lage einen oder mehrere Bots anzuweisen an bestimmten Projekten auf GitHub mitzuwirken. Die grobe Funktionsweise des Refactoring-Bots wird im Schaubild 3.1 verdeutlicht.

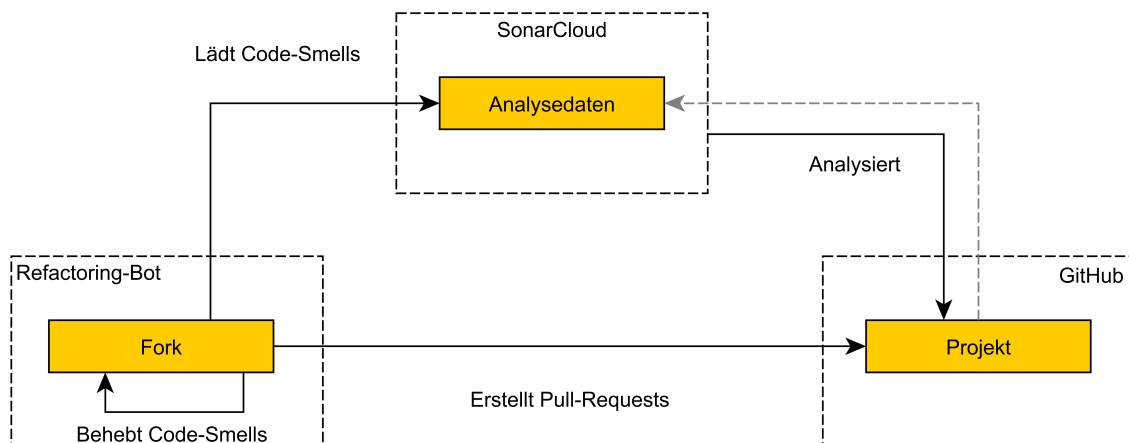


Abbildung 3.1.: Funktionsweise des Refactoring-Bots

<sup>1</sup><https://spring.io/projects/spring-boot>

<sup>2</sup><https://maven.apache.org/>

<sup>3</sup><https://www.mysql.com/de/>

### 3. Implementierung

---

Damit der Refactoring-Bot an einem Projekt mitwirken kann, muss er sich zunächst Code-Smells des Projekts beschaffen, welche vom Analysis Service SonarCloud<sup>4</sup> geliefert werden. Sobald die Code-Smells geladen sind, werden sie auf dem Fork des Bots automatisiert behoben. Im Anschluss versucht der Refactoring-Bot am Projekt mitzuwirken, indem er die Verbesserungen des Codes den Projektbesitzern vorschlägt. Das geschieht in Form eines Pull-Requests.

## 3.2. Tools und Services

Dieses Kapitel enthält die Beschreibung und Funktionalität aller Tools und Services, welche bei der Implementierung des kommentarbasierten Refactorings verwendet wurden.

### 3.2.1. Spring Tool Suite

Für die Erweiterung des Quellcodes wurde die Entwicklungsumgebung Spring Tool Suite<sup>5</sup>, kurz STS, verwendet. STS ist eine für die Entwicklung von Spring Anwendungen angepasste Eclipse-Entwicklungsumgebung. Sie kommt mit der aktuellsten Eclipse<sup>6</sup> Version und allen Spring relevanten Erweiterungen und Funktionen. Zu den Anpassungen gehört die eingebaute Erstellung von Spring Projekten, mit denen der Nutzer einfach Skelette von Spring Anwendungen generieren kann. Ein weiteres nützliches Feature der STS ist das eingebaute Spring Boot Dashboard. Mit diesem lassen sich wichtige Informationen aller Spring Anwendungen in einer separaten Ansicht anzeigen. Ebenfalls können von hier aus die Spring Anwendungen gestartet, gestoppt oder gefiltert werden [Mar15].

### 3.2.2. JGit

JGit ist eine Softwarebibliothek, welche in der Programmiersprache JAVA geschrieben ist. Sie ermöglicht die programmatische Nutzung des verteilten Versionverwaltungssystems Git. Um JGit in einem Projekt zu nutzen, muss lediglich die passende Projektabhängigkeit<sup>7</sup> zu diesem Projekt hinzugefügt werden.

### 3.2.3. Javaparser

Der Javaparser<sup>8</sup> ist ein quelloffenes Javaprojekt, welches dem Nutzer ermöglicht sowohl einzelne Javodateien, als auch komplette Javaprojekte zu parsen. Die Javodateien werden entweder direkt oder über ihre Wurzelordner eingelesen. Mit dem Javaparser ist es möglich alle Inhalte von Javodateien wie Methoden und Variablen zu untersuchen. Ebenfalls ermöglicht der Javaparser das Auflösen von Abhängigkeiten zwischen verschiedenen Javodateien. So ist es beispielsweise möglich zu einzelnen Klassen all ihre Super- und Subklassen in anderen Javodateien zu finden.

---

<sup>4</sup><https://sonarcloud.io/about>

<sup>5</sup><https://spring.io/tools>

<sup>6</sup><https://www.eclipse.org/>

<sup>7</sup><https://mvnrepository.com/artifact/org.eclipse.jgit/org.eclipse.jgit/5.3.0.201903130848-r>

<sup>8</sup><https://github.com/javaparser/javaparser>

Eine weitere wichtige Funktion vom Javaparser ist das Bearbeiten vom Quellcode der analysierten Javodateien. So ist es möglich Inhalte hinzuzufügen, oder vorhandene Inhalte zu verändern oder entfernen.

### 3.2.4. ANTLR

Das von Terrence Parr entwickelte ANTLR<sup>9</sup>, kurz für „Another Tool for Language Recognition“, ermöglicht dem Nutzer eigene, kontextfreie Grammatiken zu schreiben. Diese Grammatiken können für die Validierung von Texten oder Binärdateien genutzt werden. Mithilfe der vom Nutzer geschriebenen Grammatik kann ANTLR automatisch einen Parser und Lexer für eine unterstützte Programmiersprache generieren [PHF14].

Um ANTLR im eigenen Projekt nutzen zu können, muss das Projekt mithilfe von Maven eingerichtet werden. Dafür muss das Projekt mit passenden Projektabhängigkeiten<sup>10</sup> erweitert werden. Ergänzend zu den Projektabhängigkeiten, muss für die automatisierte Generierung der Parser- und Lexer-Klassen auch ein Plugin<sup>11</sup> hinzugefügt werden.

Nachdem das Projekt für die Arbeit mit ANTLR konfiguriert wurde, muss die Grammatik geschrieben werden. Das Schreiben der kontextfreien Grammatik erfolgt mithilfe einer vom Nutzer erstellten g4-Datei. Diese wird mit Lexer- und Parserregeln gefüllt, welche die vom Nutzer erwünschte Grammatik beschreiben [Gab17].

---

```
grammar Beispielgrammatik;

// Parser-Regeln
rechnung: ZAHL OPERATION ZAHL VERGLEICH ZAHL;

// Lexer-Regeln
ZAHL: [0-9]+;
VERGLEICH: ('=' | '!=' | '<' | '>' | '>=' | '<=');
OPERATION: ('-' | '+' | '*' | '%' | '/');
LEERZEICHEN: ' ' -> skip;
```

---

**Listing 3.1:** Beispiel für eine ANTLR-Grammatik

Wie man der Beispielgrammatik aus Listing 3.1 entnehmen kann, beginnt eine Grammatikdatei immer mit dem Schlüsselwort „grammar“, gefolgt vom Grammatiknamen. Wie bei der Programmiersprache JAVA, muss jede Zeile mit einem Semikolon beendet werden. Üblicherweise wird die Grammatikdatei so strukturiert, dass zuerst alle Parser- und dann alle Lexer-Regeln definiert werden. Parser-Regeln müssen mit Kleinbuchstaben und Lexer-Regeln mit Großbuchstaben anfangen, jedoch werden für eine bessere Übersichtlichkeit meistens alle Buchstaben klein beziehungsweise groß geschrieben [Gab17]. Die Terminale der kontextfreien Grammatik werden von einfachen Anführungszeichen umschlossen, während Nicht-Terminale keine Symbole benötigen. Neben den

<sup>9</sup><https://github.com/antlr/antlr4>

<sup>10</sup><https://mvnrepository.com/artifact/org.antlr/antlr4-runtime/4.7.1>

<sup>11</sup><https://www.antlr.org/api/maven-plugin/latest/usage.html>

### 3. Implementierung

vom Nutzer definierten Parser- und Lexerregeln stellt ANTLR auch eigene, vordefinierte Regeln zur Verfügung. Ein Beispiel ist die in der Beispiel-Grammatik enthaltene Parser-Regel „skip“, welche dafür sorgt, dass alle Leerzeichen übersprungen werden.

Bekommt ANTLR nun einen Text als Input wird dieser zunächst mittels der Lexer-Regeln in sogenannte „Tokens“ aufgeteilt. Diese Tokens beschreiben Teile des Inputs, welche verstandene Elemente einer Grammatik beschreiben [Aug18]. Diese Tokenisierung durch den Lexer wird im Schaubild 3.2 nochmal verdeutlicht.

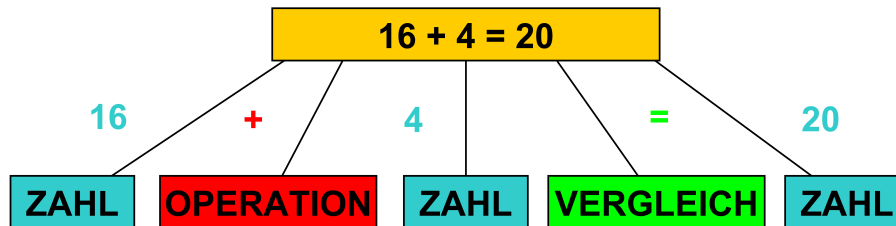


Abbildung 3.2.: Beispiel für die Funktionsweise eines Lexers

Die vom Lexer generierten Tokens werden dem Parser als Input gegeben. Dieser erstellt mittels seiner Regeln einen Syntaxbaum um die Reihenfolge der vom Lexer gelieferten Tokens zu überprüfen [Aug18]. Ist die Folge der Tokens gültig, so ist der Input in der Sprache der definierten Grammatik, sonst wird ein Fehler geworfen. Ein Beispiel für solch einen Syntaxbaum ist in Schaubild 3.3 enthalten.

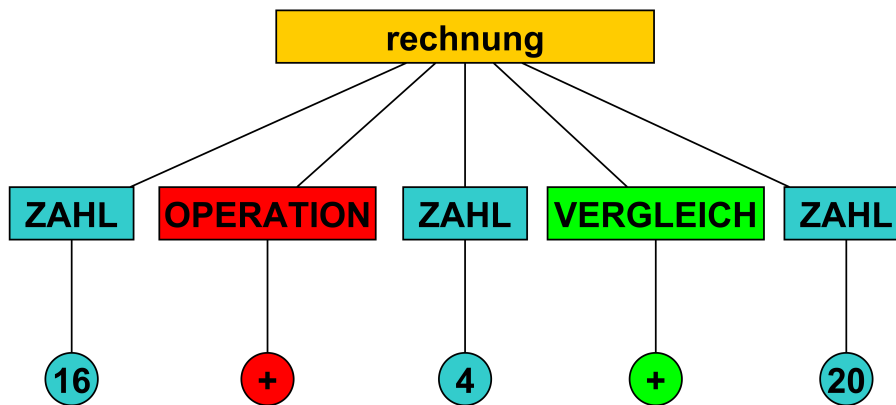


Abbildung 3.3.: Beispiel für einen Syntaxbaum eines Parsers

Im Vergleich zur Beispielgrammatik wird für das kommentargesteuerte Refactoring eine Grammatik gefordert, welche Refactoring-Befehle erkennen kann. Die Sprache der Refactoring-Grammatik sollte für den Projektverantwortlichen einfach zu verstehen, erlernen und nutzen sein. Eine allseits bekannte Sprache, welche zum Arbeiten auf Datenbanken genutzt wird ist SQL. Definiert wurde die Sprache zum ersten mal 1974 in einem Paper [DF74] von Donald Chamberlin, in dem er die Sprache zum ersten mal anhand von Beispielen definiert hat. Ihre Syntax basiert auf englischen Schlüsselwörtern, sodass SQL Befehle durch eine Aneinanderreihung von englischen Wörtern erzeugt werden können. Dadurch lässt sie sich von jedermann einfach verstehen, erlernen und nutzen.



Die Refactoring-Grammatik beschreibt eine domänenspezifische Sprache, welche sich an die Eigenschaften von SQL stützt und verschiedene Schlüsselwörter für Refactoring Operationen und Objekte verwendet. Aktuelle Schlüsselwörter für Refactoring Operationen sind „ADD“, „REMOVE“, „REORDER“ und „RENAME“, während die Menge der Refactoring Objekte mit den Schlüsselwörtern „METHOD“, „MODIFIER“, „ANNOTATION“ und „PARAMETER“ beschrieben wird. Mithilfe dieser Schlüsselwörter lassen sich alle vom Refactoring-Bot unterstützten Refactorings beschreiben.

---

```
Benenne eine Methode zu 'newName' um:
@Bot RENAME METHOD TO newName

Entferne den Parameter 'c' aus einer Methode:
@Bot REMOVE PARAMETER c

Markiere eine Methode mit einer 'Override' Annotation:
@Bot ADD ANNOTATION Override

Sortiere die Modifikatoren eines Felds oder einer Methode um:
@Bot REORDER MODIFIER
```

---

### Listing 3.2: Refactoring-Beispiele mittels ANTLR

Die im Listing 3.2 aufgeführten Refactoring-Beispiele zeigen, dass jeder Befehl mit einem „@“ Zeichen anfängt, an das der Nutzernamen des Bots angehängt wird, da das die übliche Vorgehensweise ist, um einen bestimmten Nutzer in GitHub-Kommentaren anzusprechen. Auf die Anrede des Bots folgt der eigentliche Refactoring-Befehl, welcher stark der Datenbanksprache SQL ähnelt und auf englischen Schlüsselwörtern basiert. Das erste Schlüsselwort gibt immer die Refactoring Operation wie „hinzufügen“ oder „entfernen“ an, während das zweite Schlüsselwort das zu refactornde Objekt wie eine Methode oder eine Annotation beschreibt. Im Anschluss wird mit optionalen Hilfsschlüsselwörtern oder Variablen der Befehl abgeschlossen. Zu den Variablen gehören Dinge wie Methodennamen, Parameternamen oder Namen von Annotationen, welche beispielsweise hinzugefügt, umbenannt oder entfernt werden sollen.

Für das Anpassen oder Erweitern der Grammatikregeln eignet sich ANTLR sehr gut, da man für neue Refactorings neue Parser- und Lexerregeln in der Grammatikdatei definieren kann. Dabei ist auch das Generieren von neuen Parser- und Lexerklassen der angepassten Grammatik mit einem Knopfdruck möglich. Sollte die Sprache in Zukunft durch neue Refactorings, oder durch das Auflockern der strikten Syntax komplexer werden, garantiert ANTLR durch die Mächtigkeit der kontextfreien Sprachen auch deutlich mehr Flexibilität [JJ12] als beispielsweise reguläre Ausdrücke, welche eine andere Möglichkeit bieten, Text mit wenig Aufwand zu untersuchen [Dan06].

#### 3.2.5. wit.ai

Wit<sup>12</sup> ermöglicht seinen Nutzern sogenannte „Wit-Anwendungen“ zu erstellen. Diese Anwendungen sind in der Lage natürliche Sprache zu verarbeiten. Dafür stellt Wit einige HTTP-Schnittstellen<sup>13</sup> zur Verfügung, welche die Kommunikation mit existierenden Wit-Anwendungen ermöglichen. Diese Schnittstellen können genutzt werden, um einen einfachen Bot in einen Chatbot umzuwandeln, welcher in der Lage ist natürliche Sprache zu verstehen. Zum Nutzen der HTTP-Schnittstellen wird eine Authentifizierung mittels eines öffentlichen Tokens benötigt, welches vom Administrator der Wit-Anwendung in den Anwendungseinstellungen generiert werden kann. Dieses Token kann in beliebig vielen unterschiedlichen Anwendungen genutzt werden, welche sich die natürliche Sprachverarbeitung der Wit-Anwendung zu Nutze machen wollen. Jede von Wit angebotene HTTP-Schnittstelle verwendet den Objekttyp JSON, um auf Anfragen zu antworten.

Das Erstellen und Benutzen von Wit-Anwendungen erfordert keine Erfahrung mit irgendwelchen Tools oder Programmiersprachen, denn sie werden nicht implementiert, sondern lediglich über eine von Wit zur Verfügung gestellte GUI trainiert. Das Training erfolgt mithilfe von Beispieltextrn und der Angabe ihrer Bedeutung. Das Angeben von Bedeutungen geschieht mittels sogenannter „Entitäten“. Unter einer Entität versteht man ein Objekt, welches die Bedeutung eines natürlichsprachlichen Textes in Form von strukturierten Daten beschreibt.

Die erste Art von Entität wird in Wit als „Trait“ bezeichnet. Ein Trait kann in einem Text genau einen seiner Werte annehmen. Dieser Wert beschreibt dabei die Bedeutung des Texts aus dem Input.



**Abbildung 3.4.:** Beispiel eines Traits

Das Schaubild 3.4 zeigt wie Wit mit dem Trait „refactoring“ erkennen kann, dass ein Satz die korrekte Sortierung von JAVA-Modifikatoren erwünscht. Auf diese Weise ist es auch möglich, jedem vom Refactoring-Bot unterstützten Refactoring einen Wert im Trait „refactoring“ zuzuweisen.

Eine weitere Art von Entität ist in Wit der „free-text“. Solche Entitäten ermöglichen es aus einem bestimmten Teil eines Textes, wie z.B. einzelnen Wörtern, eine Bedeutung zu filtern.

---

<sup>12</sup><https://wit.ai/>

<sup>13</sup><https://wit.ai/docs/http/20170307>

Test how your app understands a sentence

You can train your app by adding more examples

Is the parameter 'param' even used in this method?

<input checked="" type="radio"/> refactoring	REMOVE-PARAMETER	1.000
<input type="radio"/> refactoringString	param	0.992

+ Add a new entity

✓ Validate

**Abbildung 3.5.:** Beispiel: Erkennung durch freien Text

Wie man im Schaubild 3.5 feststellen kann, ermöglicht der Trait „refactoringString“ das Filtern von variablen Refactoring-Werten wie Methoden- oder Parameternamen. Diese Funktionalität ist essenziell für Refactorings mit variablen Eingabeparametern. Darüber hinaus zeigt das Schaubild, wie mehrere Entitäten kombiniert werden können, um ein komplexeres Refactoring in einem Text zu erkennen. Der Eingabetext und die Gesamtheit aller Entitäten wird in Wit als „Ausdruck“ oder „Beispiel“ bezeichnet. Je mehr Beispiele eine Wit-Anwendung hat, desto wahrscheinlicher ist die korrekte Verarbeitung eines natürlichsprachlichen Textes. Um die Wahrscheinlichkeit zu erhöhen, einen natürlichsprachlichen Text korrekt zu verarbeiten, muss die Wit-Anwendung trainiert werden.

Für das Training der Wit-Anwendung kann die von Wit zur Verfügung gestellte GUI genutzt werden. Für das programmatische Trainieren der Anwendung bietet Wit auch die Möglichkeit, das Training über eine HTTP-Schnittstelle durchzuführen.

Test how your app understands a sentence

You can train your app by adding more examples

I think we should call this method something like 'removeObject'.

<input checked="" type="radio"/> refactoring	RENAME-METHOD	0.987
--	---------------	-------

+ Add a new entity

✓ Validate

**Abbildung 3.6.:** Fehlerhafte Erkennung eines Textes

Schaubild 3.6 zeigt, dass eine Wit-Anwendung aus einem Text auch eine uneindeutige oder falsche Bedeutung erkennen kann. So kann beispielsweise bei einer Methodenumbenennung der neue Methodennamen fehlen oder mehrfach vorkommen. Solch eine fehlerhafte Rückgabe der Wit-Anwendung würde beim Refactoring-Bot zu Problemen führen, da dieser nicht wüsste, wie er den neuen Methodennamen wählen sollte. In diesem Fall muss ein Administrator diesen Fehler korrigieren, indem er fehlerhaft erkannte Entitäten eines Textes löscht, sie anpasst, oder fehlende Entitäten zu diesen Text hinzufügt. Im Schaubild 3.7 wird die vom Administrator verbesserte Entität dargestellt.

### 3. Implementierung



Abbildung 3.7.: Korrigierte Bedeutung eines Textes

Durch das Betätigen des „Validate“ Knopfes wird das Training der Anwendung getriggert. Nach einem erfolgreich durchgeführten Training sollte die Anwendung in Zukunft bei ähnlichen Texten besser agieren.

Eine letztes wichtiges Feature von Wit ist die sogenannte „Inbox“. Die Inbox enthält alle Anfragen, welche über die HTTP-Schnittstelle der Wit-Anwendung getätigt wurden. Die in der Inbox gespeicherten Anfragen können nicht über die HTTP-Schnittstelle ausgelesen werden, und können nur von Administratoren der Wit-Anwendung angeschaut und verarbeitet werden.

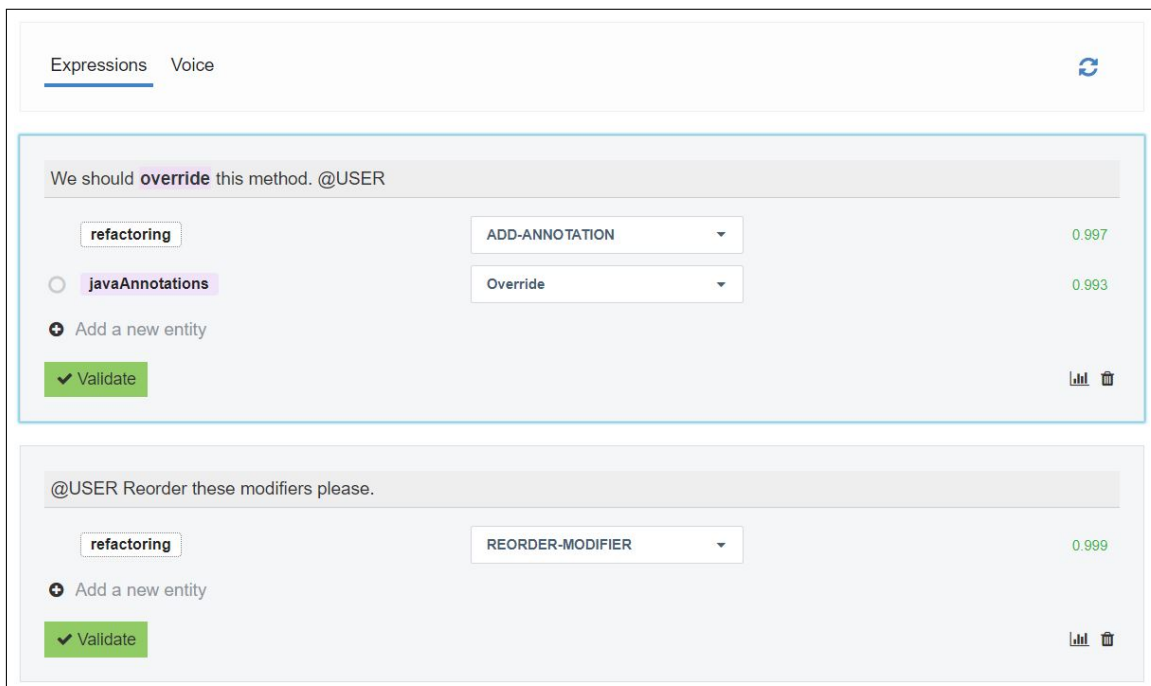


Abbildung 3.8.: Inbox einer Wit-Anwendung

Im Schaubild 3.8 ist eine Inbox mit zwei bearbeiteten Anfragen vorzufinden, welche nicht durch einen Administrator validiert wurden. Im Falle, dass die Anfrage korrekt verstanden wurde, kann ein Administrator der Wit-Anwendung diese validieren und den Ausdruck für das Training der Anwendung nutzen. Sollte der Ausdruck jedoch fehlerhaft sein, kann der Administrator diesen vor der Validierung manuell anpassen.

Die Wit-Anwendung, welche vom Refactoring-Bot verwendet wird, nutzt drei Entitäten, welche im Schaubild 3.9 dargestellt sind. Die erste und wichtigste Entität nennt sich „refactoring“. Sie erkennt alle vom Refactoring-Bot unterstützten Refactoring-Operationen. Die zweite Entität nennt sich „javaAnnotations“ und erkennt alle JAVA-Annotationen, welche aktuell in automatisierten Refactorings genutzt werden. Zuletzt gibt es die „refactoringString“ Entität, welche den veränderten Text eines Refactorings erkennt. Bei einer Umbenennung von Methoden ist das beispielsweise der neue Methodenname.

Your app uses 3 entities

Entity	Description	Values
<b>refactoringString</b> → LOOKUP STRATEGIES <b>free-text &amp; keywords</b>	User-defined entity	newerMethodName, newerName, setzeA, addieren, delete, readMe, getStats, revertStatus, reorderThat, somethingSpecial
<b>javaAnnotations</b> → LOOKUP STRATEGIES <b>free-text &amp; keywords</b>	User-defined entity	Override
<b>refactoring</b> → LOOKUP STRATEGIES <b>trait</b>	User-defined entity	REORDER-MODIFIER, RENAME-METHOD, REMOVE-PARAMETER, ADD-ANNOTATION

Abbildung 3.9.: Entitäten der Refactoring-Sprache

Wit geht im Vergleich zu ANTLR einen Schritt weiter und ermöglicht dem Nutzer natürliche Sprache zu nutzen, um mit dem Refactoring-Bot zu kommunizieren. Auf diese Weise kann sich der Nutzer so ausdrücken, wie er es am liebsten möchte und muss dabei keine Grammatiksyntax erlernen, wie es bei ANTLR der Fall ist. Darüber hinaus kann die Wit-Anwendung trainiert werden, sodass der Refactoring-Bot aus vergangenen Interaktionen lernen kann, was in der im Kapitel 2.3.4 vorgestellten Studie, von vielen Teilnehmern gefordert wurde.

### 3.3. Relevante JAVA-Objekte

In diesem Kapitel werden alle JAVA-Objekte vorgestellt, welche bei der Implementierung des kommentargesteuerten Refactorings eine wichtige Rolle spielen.

#### 3.3.1. Git-Konfiguration

Git-Konfigurationen werden vom Nutzer des Refactoring-Bots erstellt und in der Datenbank persistiert. Sie enthalten alle wichtigen Informationen über den Filehoster, das vom Bot zu refactornde Repository und den Account des Bots. Zu den Informationen des Repositories gehören

### 3. Implementierung

---

Daten wie der API-Link des Repositories und des generierten Forks, auf welchem der Bot arbeitet. Die Accountinformationen des Bots enthalten den API-Token, die Email-Adresse und den Nutzernamen. Diese werden vom Bot genutzt, um mit der API des Filehosters zu kommunizieren und um Commits mittels Git zu tätigen.

#### 3.3.2. Bot-Issue

Das Bot-Issue Objekt enthält alle wichtigen Daten, welche für das Durchführen von Refactorings benötigt werden. Dazu gehören alle Informationen des durchzuführenden Refactorings, der Dateipfad und die Codezeile des Code-Smells. Darüber hinaus enthält ein Bot-Issue auch die Pfade aller JAVA-Dateien des Projekts, sodass projektweite Refactorings, wie das Umbenennen von Methoden, durchgeführt werden können.

#### 3.3.3. Refactored-Issue

Das Refactored-Issue ist ein weiteres Objekt, welches in der Datenbank persistiert wird. Es enthält Daten, welche in der Zukunft für die Anzeige und Auswertung von Statistiken genutzt werden könnten. Unter den persistierten Informationen befindet sich eine eindeutige ID, welche sich aus dem Filehoster bzw. Analysis Service und der eindeutigen Kommentar-ID bzw. Issue-ID zusammensetzt. Mittels dieser ID kann der Refactoring-Bot ebenfalls feststellen, ob er den Issue eines Kommentars bzw. einer statischen Codeanalyse schon einmal refactored hat. Weitere im Refactored-Issue gespeicherten Informationen sind der Status, die Art und das Datum des durchgeführten Refactorings.

#### 3.3.4. Bot-Konfiguration

Die Bot-Konfiguration ist ein einzigartiges Objekt, welches global im ganzen Java-Projekt aktiv ist. Diese Konfiguration wird nicht in der Datenbank persistiert, sondern in einer Konfigurationsdatei vor dem Start der Anwendung konfiguriert. In dieser Datei können wichtige Projekteigenschaften des Refactoring-Bots eingestellt werden. Dazu gehört das Ein-/Ausschalten des eingebauten Scheduling, das Wählen des lokalen Ordners, in welchem die vom Bot geklonten Forks gespeichert werden, oder das Wählen des öffentlichen Authentifizierungstoken der Wit-Anwendung, welche für die natürliche Sprachverarbeitung von Pull-Request-Kommentaren verwendet wird.

#### 3.3.5. Bot-Pull-Request

Der Refactoring-Bot verwendet ein einheitliches Pull-Request-Objekt. Dieses Objekt enthält alle für den Refactoring-Bot wichtige Daten, welche in einem Pull-Request enthalten sind. Dazu gehört beispielsweise eine eindeutige ID, der Titel des Pull-Requests, der Nutzernamen des Erstellers und die Namen des Ursprungs- und Ziel-Branche.

### 3.3.6. Bot-Pull-Request-Kommentar

Passend zum einheitlichen Pull-Request existiert auch ein Objekt für dessen Zeilenkommentare. Dieses enthält unter anderem Daten wie eine eindeutige ID, die enthaltene Nachricht und den Nutzernamen des Erstellers, den Dateipfad und die Zeile, die zusammen beschreiben, an welcher Stelle im Code sich der Kommentar befindet. Die letztere Information ist für das Durchführen von Refactorings essenziell, da sie dem Refactoring-Bot ermöglicht, den im Kommentar erwähnten Code-Smell im Quellcode zu finden.

## 3.4. Plattformunabhängige Datensammlung

Bei der Implementierung des kommentargesteuerten Refactorings wurde darauf geachtet, dass in Zukunft neben GitHub weitere Filehoster unterstützt werden können. Für diesen Zweck wurden einheitliche Models für Pull-Request und Kommentar-Objekte eingeführt, welche vom Refactoring-Bot genutzt werden. Diese beiden Models enthalten dabei nur Daten, welche bei der Durchführung des kommentargesteuerten Refactorings essenziell sind. Damit ein Filehoster vom Refactoring-Bot genutzt werden kann, müssen seine Daten zunächst in das einheitliche Model übersetzt werden. Diese Übersetzung erfolgt mit einem Filehoster spezifischen Übersetzer, welcher das Model des Filehosters ins einheitliche, vom Bot verstandene Model übersetzt.

Für die Unabhängigkeit von Plattformen muss ebenfalls beachtet werden, dass verschiedene Filehoster, verschiedene Schnittstellen anbieten. Diese müssen, genauso wie deren Models, im Refactoring-Bot vereinheitlicht werden. Dazu muss für jeden Filehoster ein „filehoster data grabber“ implementiert werden, welcher alle für den Refactoring-Bot essenziellen Endpoints der Filehoster-API anspricht. Im Anschluss muss diese Klasse von einer einheitlichen Schnittstelle des Bots aufgerufen werden. Dieser einheitliche „data grabber“ hat die Funktion, alle Refactoring-Anfragen an passende Filehoster-APIs weiterzuleiten und dafür zu sorgen, dass die eingehenden Daten vom passenden Übersetzer verarbeitet werden. Die Funktionsweise der plattformunabhängigen Datensammlung wird im Schaubild 3.10 veranschaulicht.

### 3. Implementierung

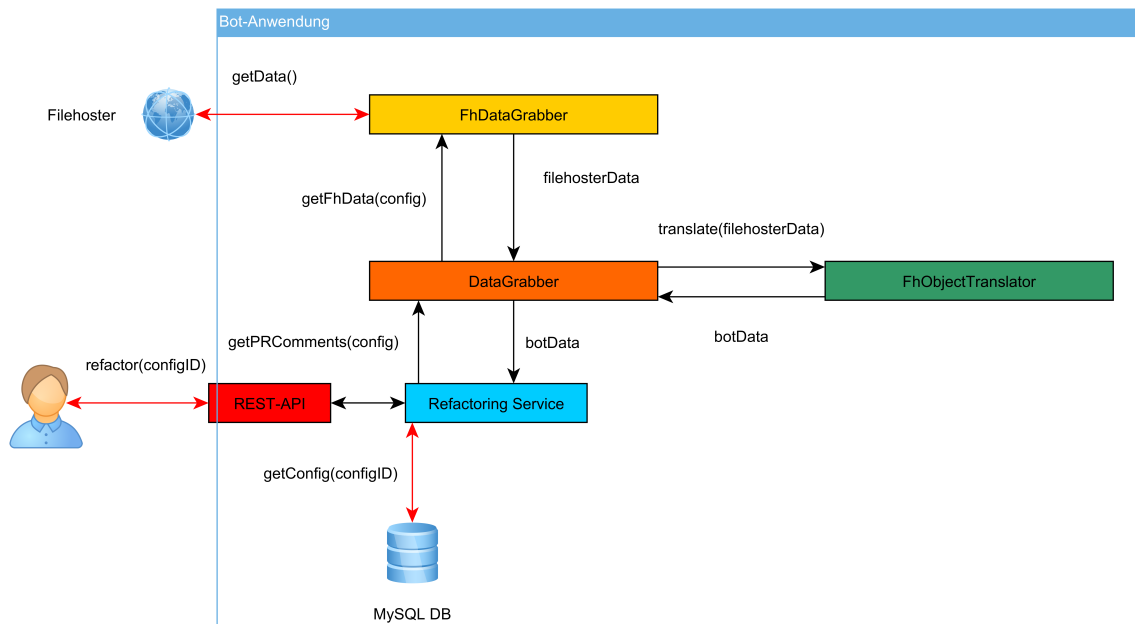


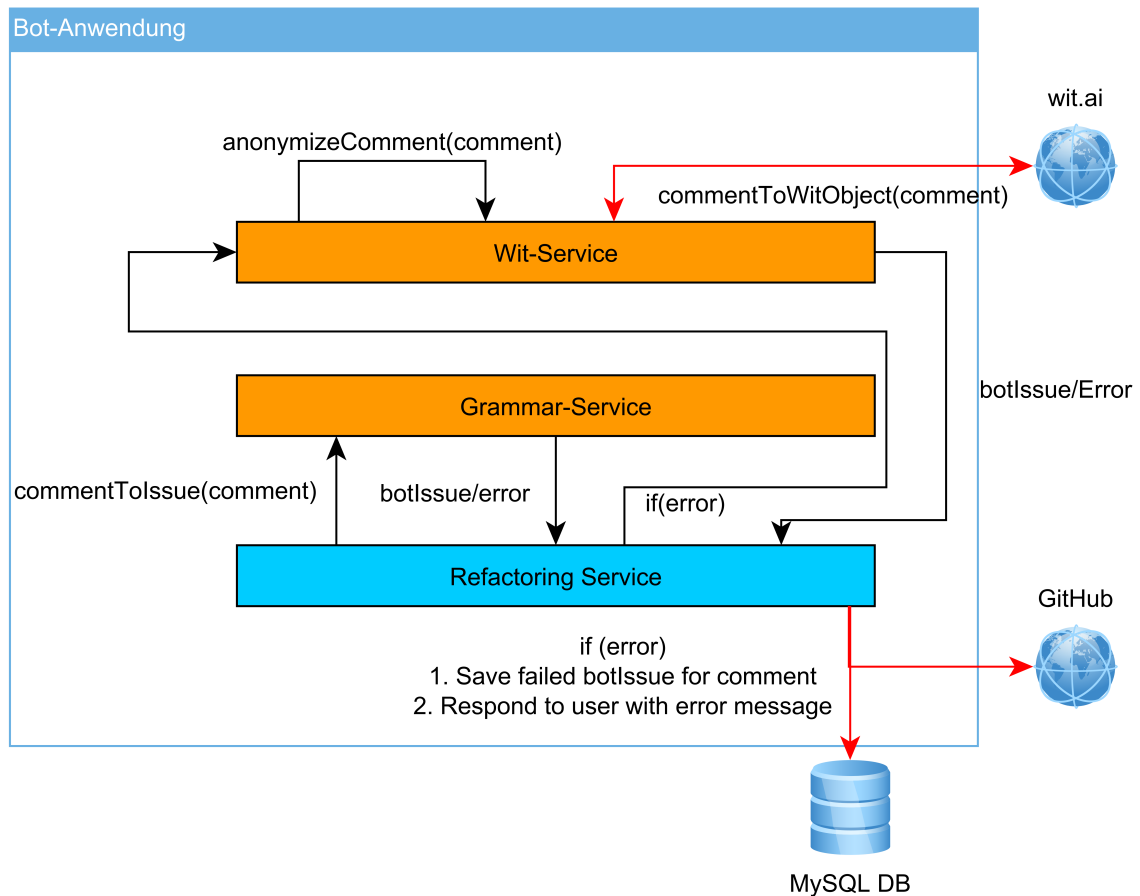
Abbildung 3.10.: Plattformunabhängiges Sammeln der Pull-Request-Kommentare

Der Prozess des kommentargesteuerten Refactorings wird durch einen Nutzer initiiert, welcher über die REST-Schnittstelle des Refactoring-Bots seinen Refactoring-Service aufruft, welcher für das Durchführen der Refactorings zuständig ist. Der Aufruf erfolgt mittels einer ID, welche das Auslesen der passenden Git-Konfiguration aus der Datenbank ermöglicht. Mithilfe der Daten dieser Konfiguration wird die einheitliche Schnittstelle aufgerufen, welche die Anfrage an die passende API des passenden Filehosters weiterleitet. Im Anschluss wird der passende Übersetzer genutzt, um die Rückgabe der API ins einheitliche Modell des Refactoring-Bots zu übersetzen. Die übersetzten Daten werden dem Refactoring-Service zurückgegeben, welcher den Refactoring-Prozess mit den übersetzten Pull-Requests und deren Kommentaren fortsetzen kann.

### 3.5. Kommentarverarbeitung

Für die Kommentarverarbeitung werden zunächst alle vom Bot gesammelten und übersetzten Pull-Requests und deren Kommentare im Refactoring-Service gesammelt. Diese Pull-Requests werden zunächst alle vom Bot analysiert. Dabei werden nur Pull-Requests beachtet, welche zum Bot-Account aus der Git-Konfiguration gehören. Im nächsten Schritt werden alle Kommentare der gefilterten Pull-Requests durchsucht. Dabei werden nur Kommentare beachtet, welche von anderen Nutzern erstellt und nicht in Vergangenheit schon einmal bearbeitet wurden. Um das zu überprüfen, kann der Refactoring-Bot auf die Datenbank zugreifen und prüfen, ob ein Refactored-Issue mit gleicher ID schon existiert.





**Abbildung 3.11.:** Übersetzung der Kommentare

Das Schaubild 3.11 demonstriert den Verarbeitungsprozess, welcher mit Kommentaren durchgeführt wird, die alle beschriebenen Bedingungen erfüllen. Dafür wird ein Kommentar zunächst an den Grammar-Service weitergeleitet. Dieser nutzt die ANTLR-Grammatik, um diesen auf Gültigkeit zu überprüfen. Gehört der Kommentar zu der Sprache der ANTLR-Grammatik, so wird aus den Kommentar-Daten ein Bot-Issue erzeugt und an den Refactoring-Service zurückgegeben. Gehört der Kommentar nicht zur Sprache der ANTLR-Grammatik, wird ein Fehler geworfen, der im Refactoring-Service gefangen wird. Dieser Fehler löst den Aufruf des Wit-Service aus, welcher die natürliche Sprachverarbeitung implementiert.

Der Wit-Service anonymisiert zunächst potentiell sensible Daten innerhalb des Kommentars und sendet im Anschluss eine Anfrage an die externe Wit-API, welche die Bedeutung des Texts in Form von einem Wit-Objekt zurückgibt. Lässt sich dieses Wit-Objekt vom Wit-Service eindeutig auf ein vom Bot unterstütztes Refactoring abbilden, so wird aus dem Wit-Objekt ein Bot-Issue erzeugt und dem Refactoring-Service zurückgegeben. Schlägt die Kommentarverarbeitung auch im Wit-Service fehl, wird im Refactoring-Service ein fehlgeschlagenes Refactored-Issue in die Datenbank gespeichert, damit der Refactoring-Bot nicht wiederholt versucht, einen nicht refactorbaren Kommentar zu verarbeiten. Im Anschluss antwortet der Refactoring-Bot auf den betroffenen Kommentar mit einer passenden Fehlernachricht.

### 3. Implementierung

Erhält der Refactoring-Service ein Bot-Issue eines erfolgreich verarbeiteten Kommentars, kann der Refactoring-Prozess fortgeführt werden.

#### 3.6. Durchführung der automatisierten Refactorings

Nachdem ein Kommentar erfolgreich zu einem Bot-Issue verarbeitet wurde, kann der Refactoring-Service generisch die passende Refactoring-Klasse mit dem Bot-Issue aufrufen, um das zum Bot-Issue passende Refactoring durchzuführen. Der Refactoring-Prozess wird im Schaubild 3.12 demonstriert.

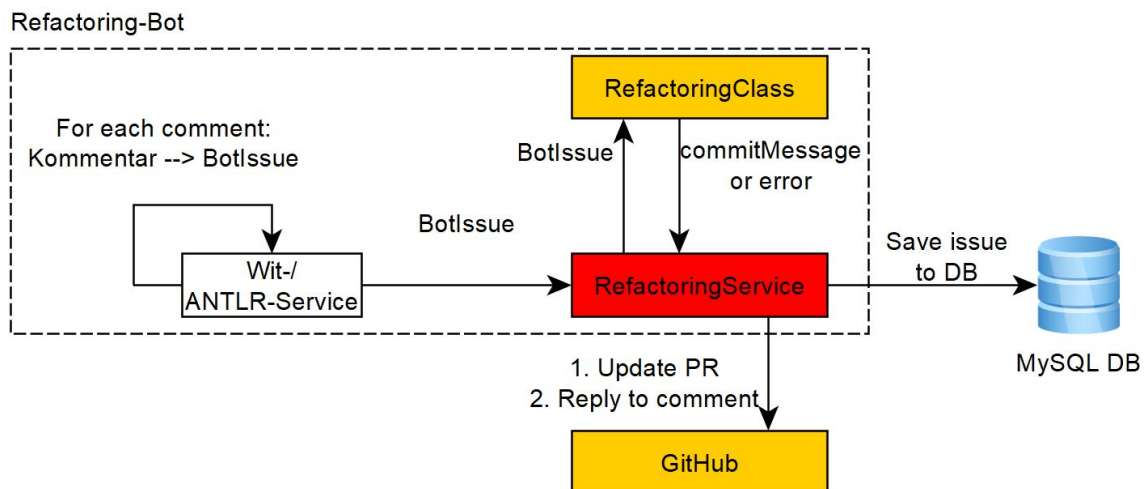


Abbildung 3.12.: Workflow des automatisierten Refactorings

Die Refactoring-Klasse führt das Refactoring aus und gibt bei Erfolg eine passende Commit-Nachricht zurück. Diese Commit-Nachricht wird genutzt um die Codeänderungen auf GitHub zu pushen und den Pull-Request zu aktualisieren. Dies geschieht mittels der JAVA-Bibliothek JGit, welche das programmatische Ausführen von Git-Befehlen ermöglicht. Daraufhin nutzt der Refactoring-Bot die GitHub-API, um den Nutzer mittels einer Antwort über den Erfolg des Refactorings zu berichten. Zu guter Letzt wird aus dem Bot-Issue und den Kommentar-Daten ein Refactored-Issue erstellt, welches in der Datenbank persistiert wird. Das dient dazu, dass der Refactoring-Bot nicht noch einmal versucht, den gerade refactorten Kommentar zu verarbeiten.

Wie bei der Kommentarverarbeitung wird in Problemfällen ein fehlgeschlagenes Refactored-Issue in der Datenbank persistiert und der Nutzer wird in Form eines Kommentars über das Problem benachrichtigt.

## 3.7. Implementierung neuer Refactorings

Die aktuelle Implementierung des Refactoring-Bots nutzt den Javaparser um zwei automatisierte Refactorings umzusetzen. Das erste Refactoring ermöglicht dem Nutzer das Hinzufügen von fehlenden „Override“ Annotationen an Methoden, während das zweite Refactoring die Modifikatoren von Variablen oder Methoden korrekt sortiert.

Um die Funktionalität des Refactoring-Bots und die damit verbundenen Kommunikationsmöglichkeiten mit dem Bot zu erweitern, wurden weitere Refactorings implementiert. In diesem Unterkapitel werden zwei neue Refactorings, sowie deren Funktionsweise, beschrieben.

### 3.7.1. Umbenennen von Methoden

Das erste neue Refactoring ermöglicht dem Bot das automatisierte Umbenennen von Methoden. Dafür müssen im ersten Schritt zunächst alle Javodateien des Projekts mit dem Javaparser geparsed werden. Dann wird in der Datei des Bot-Issues die Methode gesucht, welche umbenannt werden soll. Sollte sie existieren wird ein Abhängigkeitsbaum mit allen direkten und indirekten Super- und Subklassen der Klasse mit der besagten Methode aufgebaut. Im Anschluss wird in jeder Klasse im Abhängigkeitsbaum nach Methoden mit gleicher Signatur gesucht. Diese Methoden stehen alle in Verbindung mit der Methode des Bot-Issue und müssen daher auch umbenannt werden. Sind alle abhängigen Methoden gefunden, so wird in jeder Klasse mit solch einer Methode geprüft, ob nach einer temporären Umbenennung der Methode eine Methodensignatur doppelt vorkommen würde. Ist dies der Fall, wird ein Fehler geworfen, da das Umbenennen dieser Methode zu Fehlern führen würde. Tritt der Fehlerfall nicht auf, werden zu allen Methoden ihre Methodenaufrufe gesucht. Zum Schluss werden alle Methoden und Methodenaufrufe mithilfe des Javaparsers umbenannt.

### 3.7.2. Entfernen von Methodenparametern

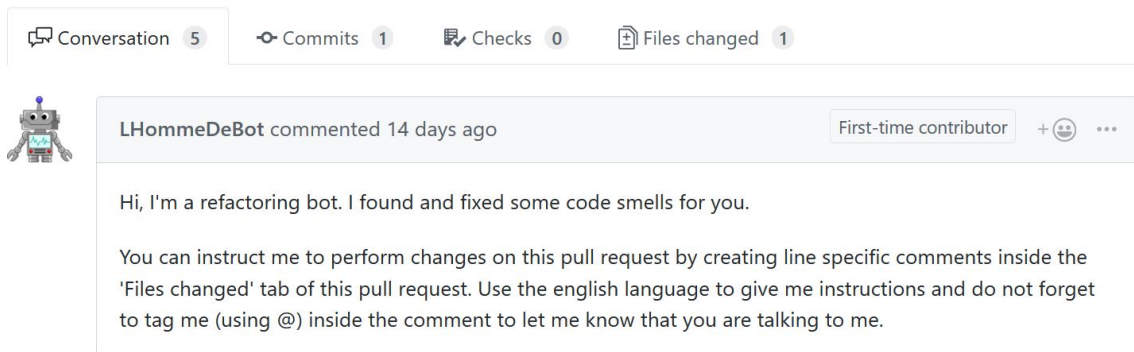
Das automatisierte Entfernen von Methodenparametern ermöglicht dem Bot unbenutzte Parameter einer Methode zu entfernen. Die Implementierung des neuen Refactorings verläuft größtenteils identisch zum Refactoring aus Kapitel 3.7.1, mit dem Unterschied, dass hier die Methode nicht umbenannt wird, sondern einer ihrer Parameter entfernt wird. Eine weitere Ergänzung ist hier, dass in allen verwandten Methoden auch noch geprüft wird, ob der zu entfernende Parameter benutzt wird. Sollte dieser Fall eintreffen wird ein passender Fehler geworfen.

## 3.8. Anleitung und Feedback

Um das im Kapitel 2.3.4 erwähnte Problem zu verhindern, dass ein Nutzer nicht versteht, wie der Refactoring-Bot genutzt werden kann, wurden die vom Bot erstellten Pull-Requests mit einer Anleitung versehen. Das Schaubild 3.13 zeigt einen Pull-Request, welcher solch eine Anleitung enthält.

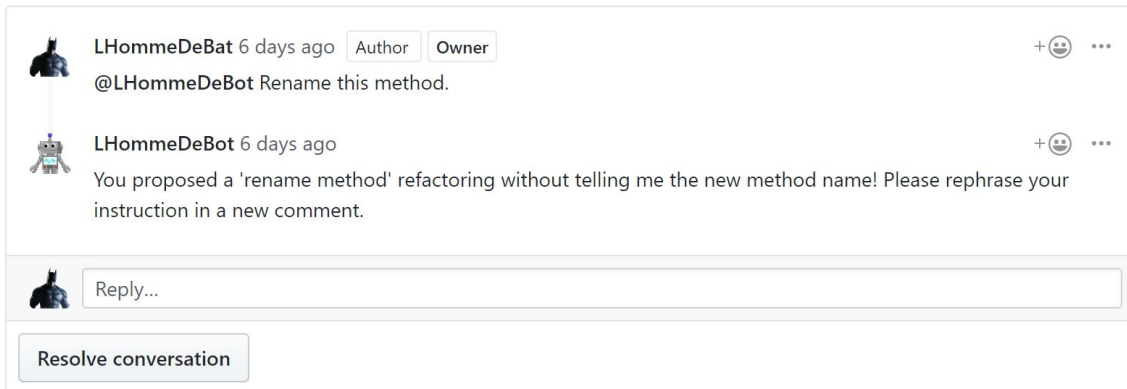
### 3. Implementierung

---



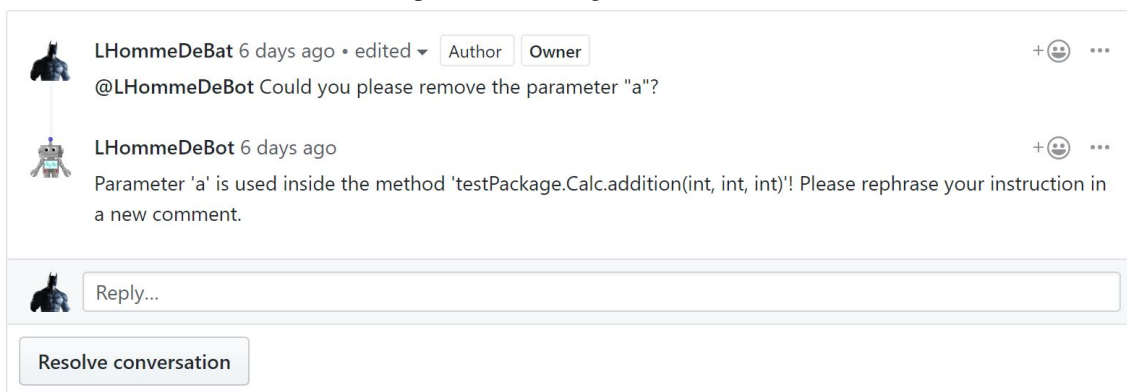
**Abbildung 3.13.:** Beispiel einer Nutzungsanleitung

Neben der Anleitung wurde der Refactoring-Bot um eine Feedback-Funktion erweitert, welche in Fehlerfällen getriggert wird. Der Refactoring-Bot gibt dabei sein Feedback in Form von GitHub-Kommentaren ab. Sollte während der Kommentarverarbeitung oder des automatisierten Refactorings ein Fehler auftreten, nutzt der Refactoring-Bot die geworfene Exception um mit ihrer Hilfe eine Nachricht aufzubauen, um den Nutzer auf GitHub über den Fehler zu informieren. Dieser vom Bot erstellte Kommentar enthält dabei nicht nur die Beschreibung des Fehlers, sondern auch einen Hinweis, wie dieser behoben werden könnte. Diese Funktionalität soll dem Problem der im Kapitel 2.3.4 vorgestellten Studie entgegenwirken, bei der sich einige Entwickler darüber beschwert haben, dass Bots schlechtes oder unverständliches Feedback geben. Im Schaubild 3.14 sind Beispiele von Feedback-Kommentaren des Refactoring-Bots enthalten.



The screenshot shows a GitHub conversation. The first message is from user LHommeDeBat, marked as 'Author' and 'Owner', posted 6 days ago. The message says '@LHommeDeBot Rename this method.' The second message is from LHommeDeBot, posted 6 days ago, replying: 'You proposed a 'rename method' refactoring without telling me the new method name! Please rephrase your instruction in a new comment.' Below the messages is a 'Reply...' input field and a 'Resolve conversation' button.

**(a)** Beispiel für einen ungenauen Kommentar



The screenshot shows a GitHub conversation. The first message is from user LHommeDeBat, marked as 'Author' and 'Owner', posted 6 days ago and marked as 'edited'. The message says '@LHommeDeBot Could you please remove the parameter "a"?' The second message is from LHommeDeBot, posted 6 days ago, replying: 'Parameter 'a' is used inside the method 'testPackage.Calc.addition(int, int, int)! Please rephrase your instruction in a new comment.' Below the messages is a 'Reply...' input field and a 'Resolve conversation' button.

**(b)** Beispiel für ein unmögliches Refactoring

**Abbildung 3.14.:** Beispiele von Fehlernachrichten



## 4. Evaluation

Für die Evaluation der implementierten Lösung des kommentargesteuerten Refactorings soll eine qualitative Studie durchgeführt werden. Ihre Ziele, Vorbereitung und Durchführung werden in diesem Kapitel beschrieben.

### 4.1. Ziele

In dieser Studie soll geprüft werden, wie gut die aktuelle Implementierung des kommentargesteuerten Refactorings in der Praxis von den Nutzern aufgenommen wird. Dabei soll vorrangig geprüft werden, ob die Nutzer mit der Bedienung des Refactoring-Bots zurecht kommen. Damit ist gemeint, dass ein Nutzer versteht, an welcher Stelle im Pull-Request, welche Art von Kommentar geschrieben werden muss, wie dieser Kommentar aufgebaut sein muss und was zu tun ist, falls ein Refactoring fehlschlägt.

Neben der Bedienung soll in der Studie auch die Qualität der natürlichen Sprachverarbeitung für die aktuell unterstützten Refactorings geprüft werden. Dabei soll speziell darauf geachtet werden, wie viele Befehle des Nutzers vom Bot verstanden werden und ob diese zu einem korrekt durchgeführten Refactoring führen.

Zuletzt sollen mithilfe der Studie weitere, für den Refactoring-Bot relevante Informationen gesammelt werden, welche die künftige Entwicklung des Refactoring-Bots beeinflussen könnten. Zu solchen Informationen gehören Änderungs- oder Verbesserungsvorschläge der aktuellen Implementierung oder Wünsche für künftige Features und Erweiterungen.

### 4.2. Vorbereitung

Damit die Studie durchgeführt werden kann, müssen zunächst Pull-Requests erstellt werden, welche in den geänderten Codeabschnitten weitere Code-Smells enthalten, die vom Refactoring-Bot behoben werden können. Das ist essenziell für das Durchführen der Studie, da nur in geänderten Codeabschnitten eines Pull-Requests Zeilenkommentare getätigt werden können.

Dafür wurde ein simples JAVA-Projekt<sup>1</sup> erstellt, bei welchem sich immer zwei verschiedene Code-Smells in unmittelbarer Nähe voneinander befinden. Sollte der Refactoring-Bot einen Code-Smell beheben, würde sich der andere im veränderten Codeabschnitt des Pull-Requests befinden, und wäre damit kommentierbar. Darüber hinaus wurde das „extract method“ Refactoring innerhalb des Projekts manuell simuliert. Dieses ist aktuell im Rahmen einer anderen Arbeit in Entwicklung.

---

<sup>1</sup><https://github.com/LaPersonneDeTest/TestProjekt>

Der Zweck dieses Refactorings ist es, aus einer großen Methode bestimmte Teile zu nehmen, und diese in eine neue, zufällig benannte Methode zu schieben, um die Komplexität der großen Methode zu verringern. Der Name dieser neuen, zufällig benannten Methode wäre dabei ein Code-Smell, der durch das Beheben des „too long method“ Code-Smells entstehen würde, und durch ein kommentarbasierendes Refactoring behoben werden müsste.

Für die Durchführung der Studie wird für jede Testperson eine Kopie des Testprojekts mitsamt drei Pull-Requests<sup>2</sup> erstellt, welche jeweils einen speziellen Code-Smell besitzen, welcher durch Zeilenkommentare markiert, und damit auch behoben werden kann. Darüber hinaus wird das vom Refactoring-Bot genutzte Scheduling auf eine Minute Verzögerung gestellt, sodass der Refactoring-Bot in kurzen Abständen die Kommentare der Testpersonen automatisiert bearbeiten kann. Das sorgt dafür, dass die Testperson nicht lange warten muss, um Feedback zu bekommen.

Da der aktuelle Refactoring-Bot nur vier von sehr vielen potenziell möglichen Refactorings unterstützt, wurde ein Aufgabenblatt erstellt, welches die drei Pull-Requests mitsamt ihrer Code-Smells auflistet und die Testperson beauftragt, genau diese, und keine anderen, zu beheben. Dies soll verhindern, dass die Testperson versucht Refactorings durchzuführen, welche von der aktuellen Version des Refactoring-Bots nicht unterstützt werden. Das Aufgabenblatt ist im Appendix B dieser Arbeit angehängt.

Zuletzt wurde für die Auswertung der Ergebnisse ein Erfahrungsbogen mit verschiedensten Aussagen über die im Kapitel 4.1 erwähnten Ziele erstellt. Der Erfahrungsbogen verwendet dabei eine Likert-Skala mit einer geraden Anzahl von Antwortmöglichkeiten, sodass eine Dichotomisierung der gesammelten Informationen möglich ist. Genau wie das Aufgabenblatt, ist der Evaluationsbogen im Appendix C vorzufinden.

### 4.3. Durchführung

Für die Durchführung der qualitativen Studie werden neun Testpersonen gewählt, welche die Grundlagen der Programmiersprache JAVA beherrschen. Das soll sicherstellen, dass keine Probleme bei der Erkennung und Behebung der Code-Smells auftreten. Um einen Projektverantwortlichen auf GitHub simulieren zu können, sollten die gewählten Testpersonen auch genug Erfahrung mit GitHub haben.

Im ersten Schritt bekommt die Testperson eine Einverständniserklärung, welche gelesen und unterschrieben werden soll. Diese ist im Appendix A dieser Arbeit angehängt. Im Anschluss wird der Testperson der Refactoring-Bot vorgestellt und der Zweck der Studie erläutert.

Nach der Einführung und möglichen Fragen bekommt die Testperson das im Kapitel 4.2 erwähnte Aufgabenblatt und maximal zehn Minuten, um die drei Pull-Requests erfolgreich zu bearbeiten. In diesen zehn Minuten hat die Testperson nur das Aufgabenblatt und die Bedienungsanleitung des Bots, welche sich in den Pull-Requests befindet, als Hilfsmittel.

Nach dem Ablauf der Bearbeitungszeit bekommt die Testperson einen Evaluationsbogen, welchen sie ausfüllen soll. Zum Abschluss wird ein kurzes Interview geführt, in welchem die Testperson die Möglichkeit bekommt, spezifischeres Feedback zum Refactoring-Bot abzugeben. Das Interview

---

<sup>2</sup><https://github.com/LaPersonneDeTest/TestProjekt/pulls>



wird mit der Frage eingeleitet, wie der Befragte die Erfahrung mit dem Refactoring-Bot empfunden hat. Des Weiteren werden im Interview auch Verbesserungs- bzw. Erweiterungsvorschläge oder Probleme diskutiert, welche der Testperson während Nutzung des Refactoring-Bots aufgefallen sind.



## 5. Ergebnisse

In diesem Kapitel werden die Ergebnisse der durchgeführten Evaluation vorgestellt.

### 5.1. Testpersonen

Die neun Testpersonen, die an der Studie teilgenommen haben, waren bis auf eine Ausnahme alle Studenten der Universität Stuttgart. Die Ausnahme bildet ein Mitarbeiter einer Firma, die sich mit Softwareentwicklung beschäftigt. Bei den acht Studenten handelt es sich um drei Softwaretechniker, drei Informatiker und zwei Mechatroniker. Das Fachsemester der untersuchten Studenten befindet sich dabei stets zwischen dem vierten Semester des Bachelors und dem dritten Semester des Masters.

Das Schaubild 5.1 zeigt die Menge an JAVA-Erfahrung, welche die Testpersonen besitzen.

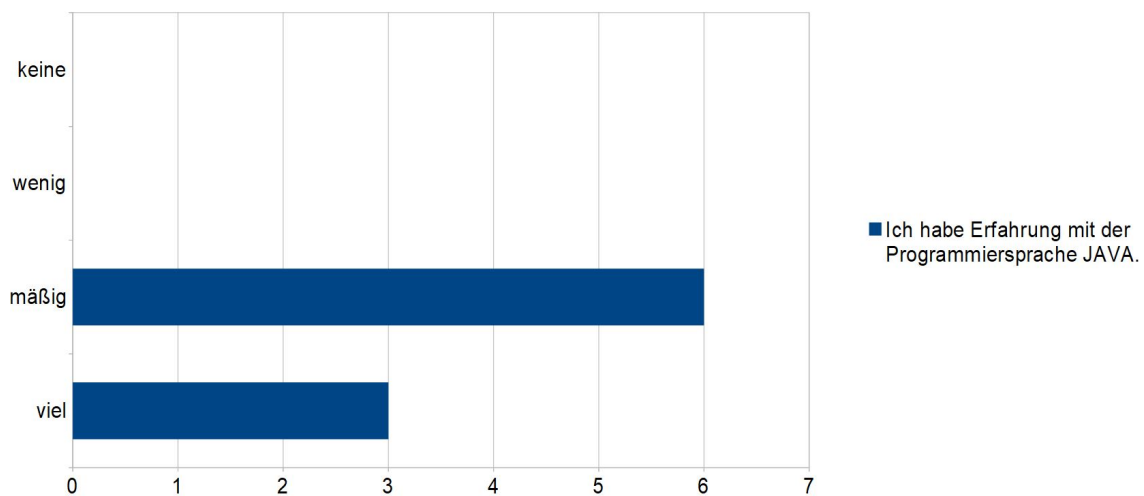
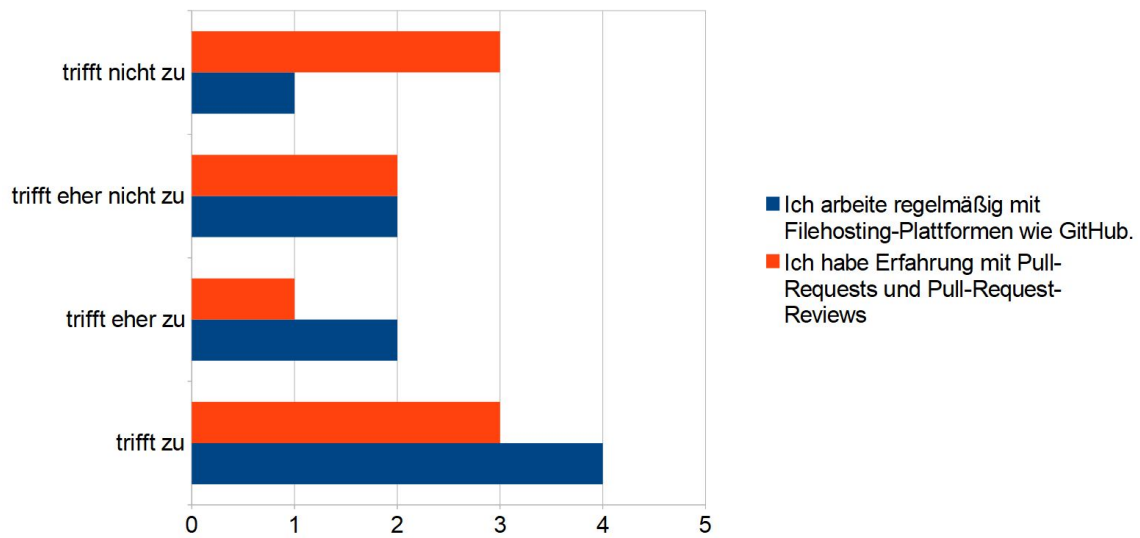


Abbildung 5.1.: Erfahrung mit JAVA

Im Vergleich dazu zeigt das Schaubild 5.2 wie regelmäßig die Testpersonen Filehosting-Services wie GitHub verwenden und wie viel Erfahrung sie dabei mit Pull-Requests und deren Reviews gesammelt haben.

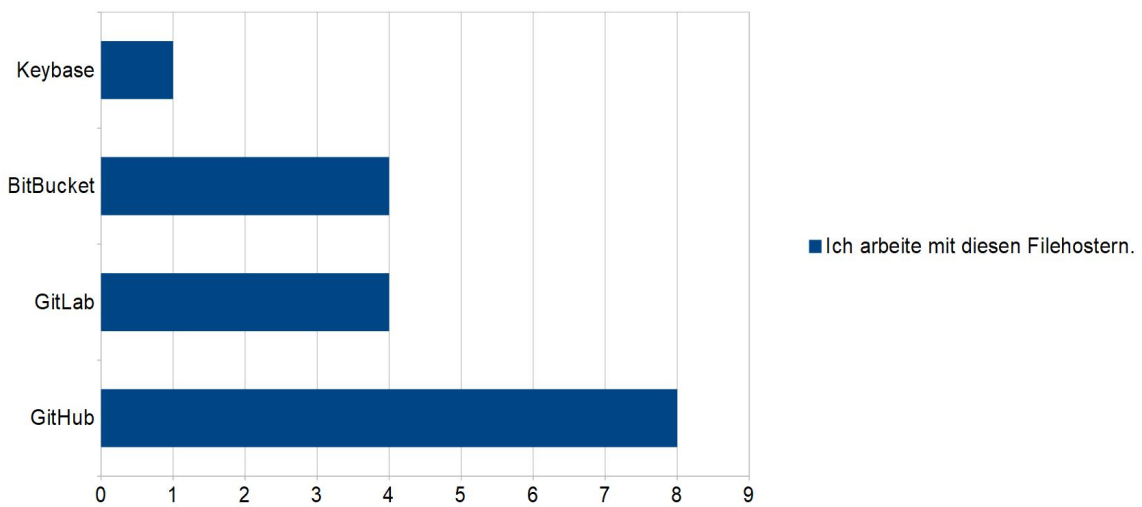
## 5. Ergebnisse

---



**Abbildung 5.2.:** Regelmäßige Nutzung von Filehostern

Zusätzlich zur regelmäßigen Nutzung von Filehostern kann man dem Schaubild 5.3 entnehmen, welche Filehosting-Services bei den Testpersonen beliebt sind.



**Abbildung 5.3.:** Genutzte Filehoster

Zum Schluss veranschaulicht das Schaubild 5.4 wie wichtig das Refactoring von Software für die Testpersonen ist.

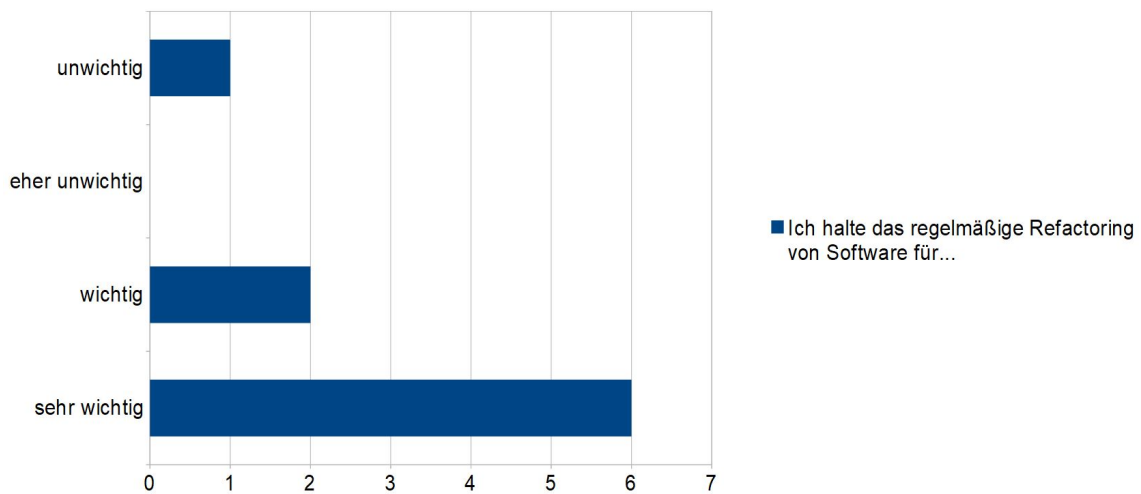


Abbildung 5.4.: Wichtigkeit von Software-Refactoring

## 5.2. Pull-Request-Aufgaben

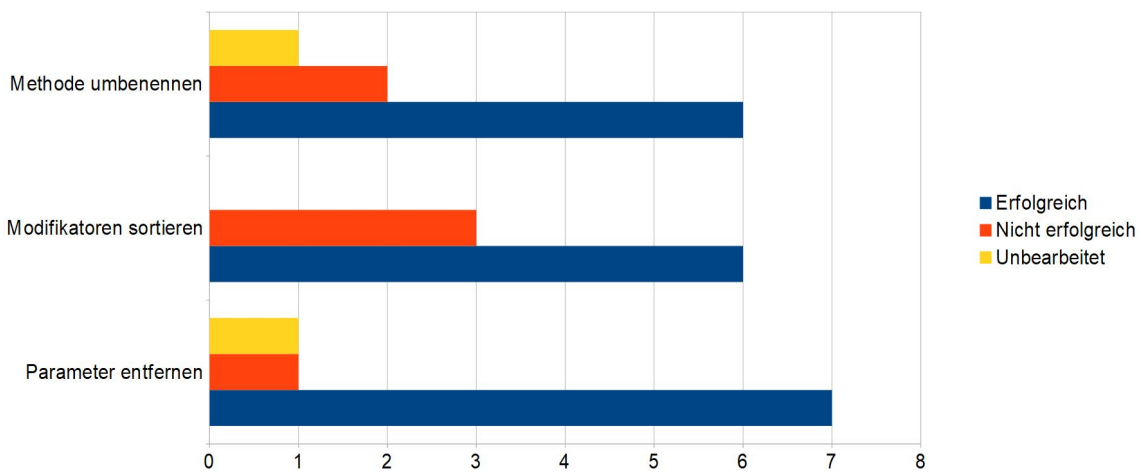


Abbildung 5.5.: Ergebnisse der Pull-Request-Bearbeitung

Das Schaubild 5.5 zeigt den Bearbeitungsstatus der drei Aufgaben, welche innerhalb von zehn Minuten von den Testpersonen bearbeitet werden sollten. Dabei wurde jede Aufgabe von mindestens sechs Testpersonen erfolgreich bearbeitet. Erfolgreich bearbeitete Aufgaben sind dabei solche, bei denen der enthaltene Code-Smell durch den Bot behoben wurde. Bei fehlgeschlagenen Aufgaben hingegen, wurde der Code-Smell trotz der vom Nutzer verfassten Refactoring-Anweisungen nicht behoben. Zum Schluss gab es zwei Testpersonen, die nicht in der Lage waren, alle drei Aufgaben im vorgegebenen Zeitraum zu bearbeiten. Unbearbeitete Aufgaben enthalten dabei keinen einzigen Kommentar, der durch die Testperson verfasst wurde.

## 5. Ergebnisse

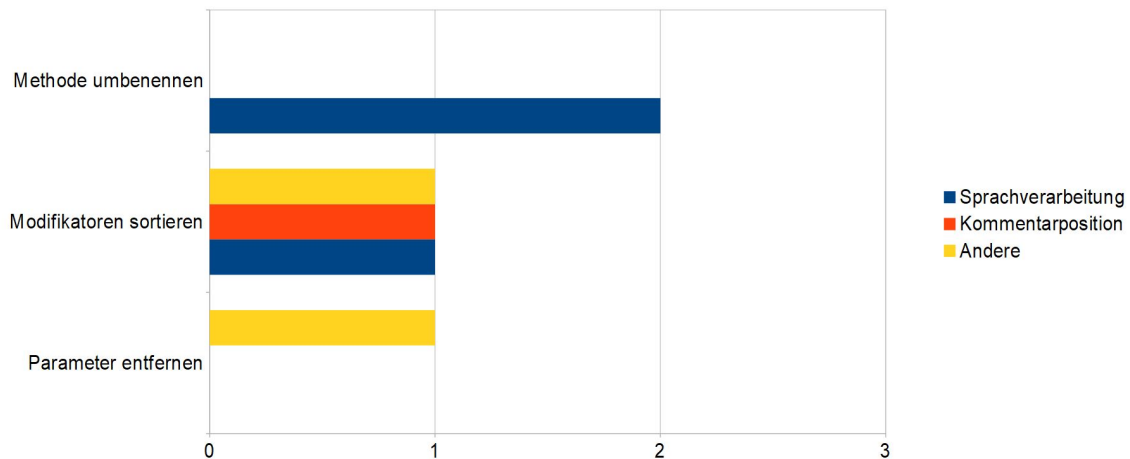


Abbildung 5.6.: Fehlergründe für nicht durchgeführte Refactorings

Die Gründe, warum einige Testpersonen bestimmte Aufgaben nicht erfolgreich bearbeiten konnten, sind im Schaubild 5.6 dargestellt. Zu sehen ist, dass der überwiegende Teil der Probleme der Spracherkennung verschuldet ist, welche den Kommentar der Testperson nicht verstehen konnte. In einem Fall wurden Kommentare an falscher Position platziert, sodass der Bot ständig versucht hat Code ohne Code-Smell zu refactorieren. Zwei Testpersonen haben jeweils versucht eine Variable bzw. eine Annotation aus der Klasse zu entfernen, was vom Refactoring-Bot aktuell nicht unterstützt wird, und im Aufgabenblatt nicht gefordert war.

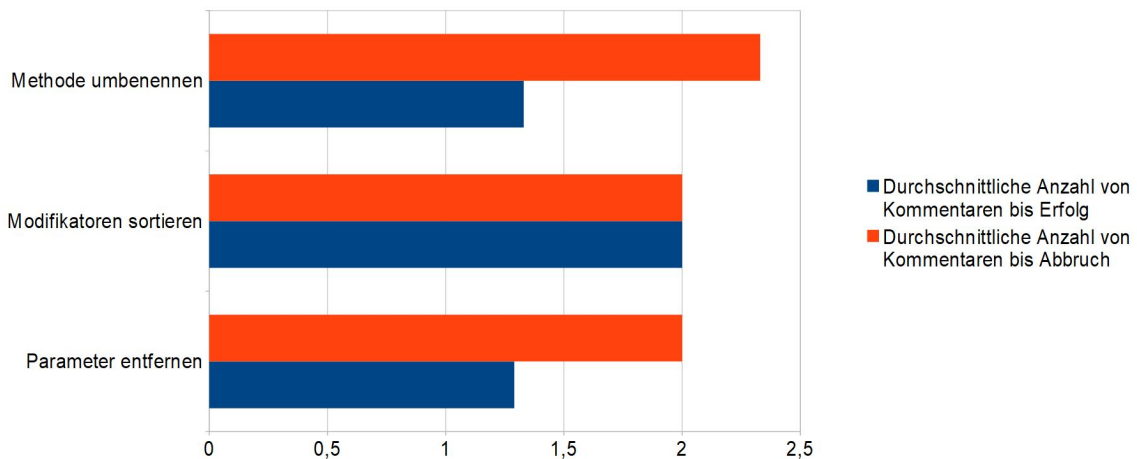


Abbildung 5.7.: Kommentar-Anzahl in bearbeiteten Pull-Requests

Die durchschnittliche Anzahl von Kommentaren in bearbeiteten Pull-Request lässt sich im Schaubild 5.7 ablesen. Dabei kann man feststellen, dass Testpersonen, die einen Pull-Request erfolgreich bearbeitet haben, im Schnitt nicht mehr als zwei Kommentare für die Bearbeitung der Aufgabe benötigt haben. Im Vergleich dazu haben Testpersonen, die die Aufgabe eines Pull-Requests nicht erfolgreich bearbeiten konnten, im Schnitt mindestens zwei Kommentare geschrieben, bevor sie die Bearbeitung abgebrochen haben.

## 5.3. Evaluationsbogen

### 5.3.1. Bedienung

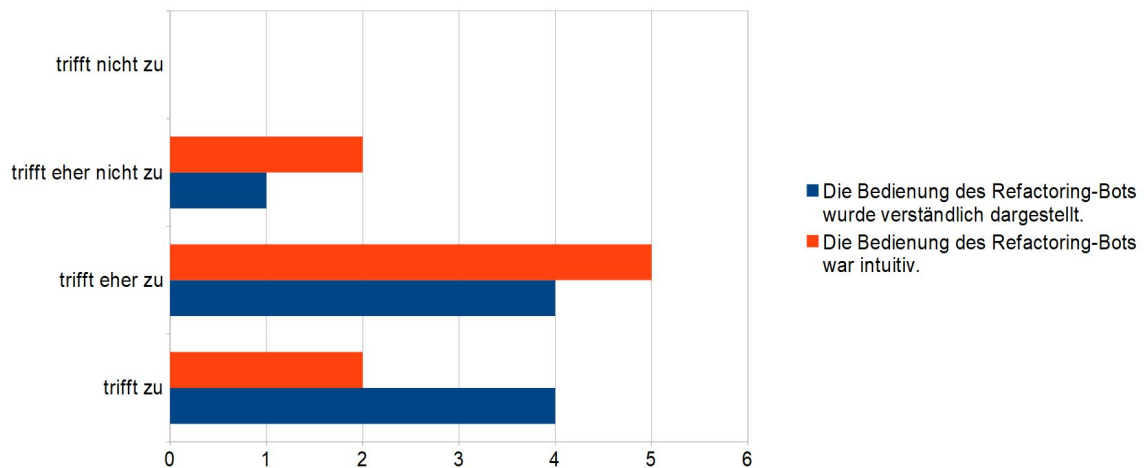


Abbildung 5.8.: Evaluationsergebnisse der Bedienung

Das Schaubild 5.8 zeigt, dass die Bedienung des Refactoring-Bots überwiegend positiv aufgenommen wurde. Mit Bedienung ist die Steuerung des Bots bzw. die Kommunikationsweise mit dem Bot gemeint, mit welcher dieser gesteuert werden kann. Das positive Feedback gilt auch für die vom Bot zur Verfügung gestellte Anleitung, welche die Bedienung des Refactoring-Bots darstellt.

Während der Interviews forderten einige Testpersonen jedoch eine detailliertere Anleitung mit Details darüber, wie ein Kommentar aussehen sollte. Als Beispiel wurden mehrmals Schlüsselwörter genannt, die im Kommentar enthalten sein müssen, damit dieser vom Bot verstanden wird. Einige Testpersonen haben auch gefordert, dass die Anleitung eine Liste von Refactorings enthält, welche vom Bot unterstützt werden. Ein Kritikpunkt, welcher von einigen Testpersonen genannt wurde, war der, dass die Anleitung zunächst eher unklar war, und erst während der Nutzung des Bots klarer wurde. Einige Testpersonen haben zunächst Teile der im Pull-Request enthaltenen Anleitung überlesen. So hat eine Testpersonen den Teil übersehen, welcher sie darauf hinweisen sollte, den Bot im Kommentar anzusprechen. Eine Person hat anfangs sogar die komplette Anleitung übersehen.

Für die Interaktion selbst haben sich einige Testpersonen gewünscht, dass auch vom Quellcode unabhängige Kommentare auf der Startseite des Pull-Requests geschrieben werden können. Eine Testperson beschwerte sich dabei über den irreführenden Prozess beim Erstellen von Review-Kommentaren. Das Problem war dabei, dass sie zunächst ein Review mit entsprechenden Kommentaren erstellt, aber nicht eingereicht hatte, sodass die Kommentare nicht vom Bot ausgelesen werden konnten. Ein weiteres Problem, welches von einer Testperson genannt wurde ist, dass die vom Bot unterstützten Zeilen-Kommentare nur an Codestellen platziert werden können, die im Pull-Request angepasst wurden. Hier besteht der Wunsch, Kommentare überall im Quellcode platzieren zu können.

### 5.3.2. Verarbeitung

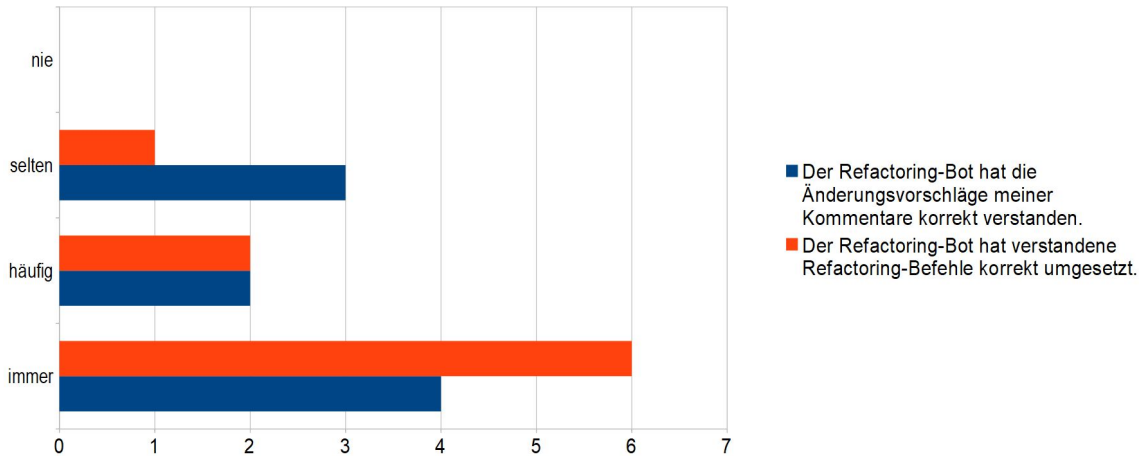


Abbildung 5.9.: Evaluationsergebnisse der Sprachverarbeitung

Dem Schaubild 5.9 lässt sich entnehmen, dass die Verarbeitung der Kommentare von den Testpersonen überwiegend positiv aufgenommen wurde. Die Probleme scheinen dabei größtenteils auf das Verständnis der Kommentare zurückzugreifen. Die Umsetzung der Refactorings durch verstandene Kommentare hingegen, ist bis auf eine Ausnahme, durchgehend positiv ausgefallen.

### 5.3.3. Feedback

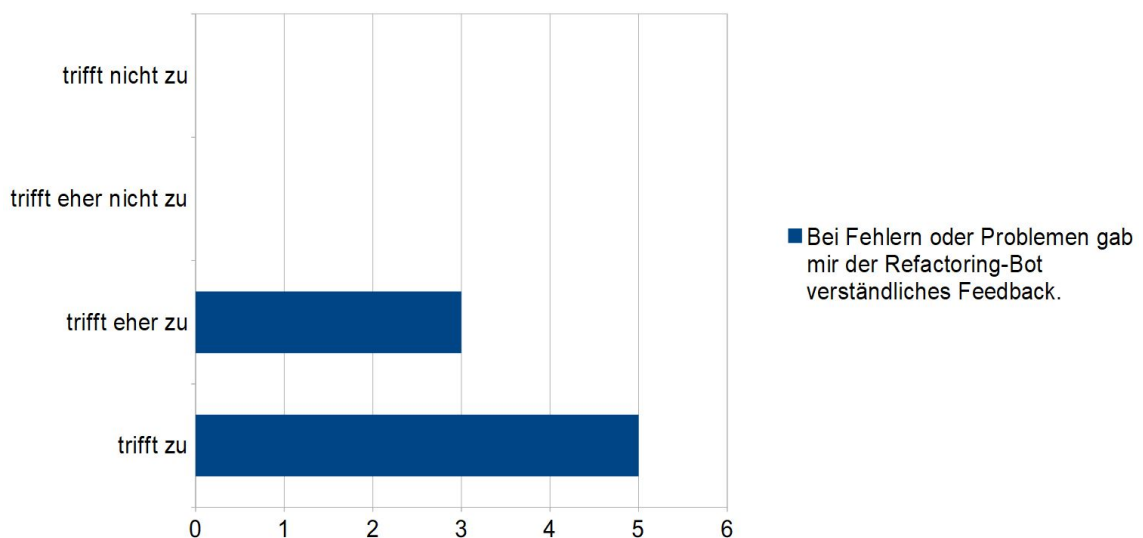
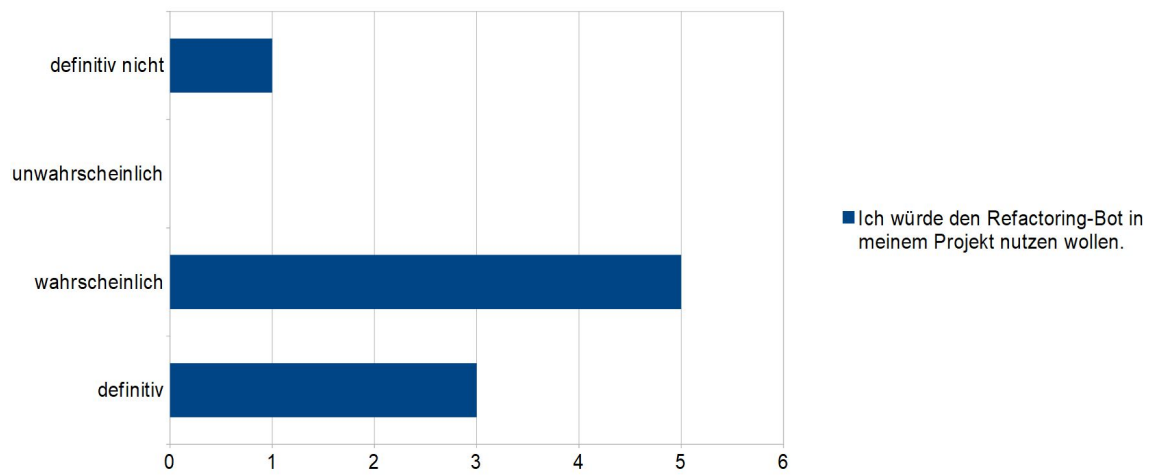


Abbildung 5.10.: Evaluationsergebnisse des Bot-Feedbacks



Das Feedback, dass der Refactoring-Bot den Nutzern in Fehlerfällen gegeben hat, ist laut Schaubild 5.10 ausschließlich positiv ausgefallen. Auffällig ist, dass eine Testperson die Aussage nicht ausgewertet hat. Diese Person hat alle Pull-Requests mit nur einem Kommentar erfolgreich bearbeiten können. In einem Fall hat eine Testperson im Interview erwähnt, dass der Refactoring-Bot in der Situation, in der er den Inhalt des Kommentars überhaupt nicht verstanden hat, kein hilfreiches Feedback gibt.

#### 5.3.4. Fazit



**Abbildung 5.11.:** Gesamteindruck vom kommentargesteuerten Refactoring

Aus dem Schaubild 5.11 lässt sich ablesen, dass die meisten Testpersonen dazu neigen würden, den kommentargesteuerten Aspekt des Refactoring-Bots in eigenen Projekten zu nutzen. Eine Testperson verneint die Verwendung des Refactoring-Bots, da dieser zu kleine Änderungen am Code vornimmt. Der besagten Testperson wäre es lieber, diese Änderungen selbst in ihrer IDE vorzunehmen.



## 6. Diskussion

In diesem Kapitel werden die Ergebnisse aus Kapitel 5 diskutiert.

### 6.1. Ergebnisse

#### 6.1.1. Mangel an Pull-Request-Erfahrung

Das Schaubild 5.2 zeigt, dass zwar alle Testpersonen ein bestimmtes Maß an JAVA-Kenntnissen besitzen, es dennoch einige Testpersonen gab, die zwar Erfahrungen mit Plattformen wie GitHub haben, diese jedoch nicht regelmäßig verwenden. Darüber hinaus kann man auch sehen, dass die regelmäßige Nutzung solcher Plattformen nicht mit der Erfahrung mit Pull-Requests und deren Reviews gleichzusetzen ist. Es lässt sich also vermuten, dass selbst auf öffentlichen Projekten einige Entwickler arbeiten, die wenig bis keine Erfahrung mit Pull-Requests haben, da sie eventuell nur die Grundfunktionen von GitHub nutzen. Die Wahrscheinlichkeit dafür ist bei Projekten mit einer großen Anzahl von Pull-Requests sicherlich kleiner, wie bei Projekten die wenige oder keine eingehenden Pull-Requests besitzen.

Eine interessante Erkenntnis ist, dass genau die Personen, die unregelmäßig mit Filehosting-Plattformen arbeiten, oder weniger Erfahrung mit Pull-Requests haben, größere Probleme mit der Bedienung des Refactoring-Bots hatten, und nicht alle Pull-Requests in vorgegebener Zeit erfolgreich bearbeiten konnten.

#### 6.1.2. Bedienungsprobleme

Die im Kapitel 6.1.1 erwähnten Testpersonen sprachen im Interview an, dass sie die Bedienungsanleitung des Refactoring-Bots nicht richtig verstehen konnten. Eine Person erwähnte speziell, dass sie zunächst nicht verstanden hat, wo und wie sie Zeilenkommentare in einem Pull-Request platzieren kann. Jedoch kann man nicht jede Schuld auf die Erfahrung der Testpersonen schieben. Selbst nach anfänglichen Schwierigkeiten haben die Personen es schlussendlich hinbekommen einige Pull-Requests erfolgreich zu bearbeiten.

Zwei dieser Testpersonen mussten jeweils einen Pull-Request unbearbeitet lassen, da ihnen die anfänglichen Verständnisprobleme zu viel Zeit weggenommen haben. Jede dieser Testpersonen hatte aber trotz geklärter Anfangsschwierigkeiten im Anschluss noch Probleme mit dem Refactoring-Bot zu interagieren. In einigen Pull-Requests dieser Testpersonen konnte der Bot die Kommentare nicht korrekt verstehen, während in anderen die Testpersonen entweder ständig Refactorings an falschen Stellen im Code, oder Refactorings, welche vom Bot aktuell nicht unterstützt werden,

verlangt haben. So hat eine Person ständig versucht schon korrekt sortierte Modifikatoren einer Variable umzusortieren. Andere Personen haben versucht, die Modifikatoren einer Variable, oder Override-Annotationen zu entfernen.

Letztlich gab es auch Probleme, die auch Nutzer mit mehr Pull-Request-Erfahrungen geplagt haben. So wurden entweder Teile, oder gar die komplette Bedienungsanleitung des Refactoring-Bots übersehen. Dies hatte zur Folge, dass die Kommunikation mit dem Refactoring-Bot gestört war, da die Kommentare nicht in der richtigen Form geschrieben wurden. So haben mehrere Testpersonen den Bot zunächst in den Kommentaren nicht angesprochen, oder gewöhnliche statt Zeilenkommentare getätigt, welche vom Bot nicht verarbeitet werden.

Um diese Probleme zu beheben, müsste die im Schaubild 3.13 gezeigte Bedienungsanleitung so angepasst werden, dass ihre Inhalte nicht so einfach zu überlesen sind. So könnte man größere Schrift, oder Aufzählungen verwenden, die beschreiben, wie ein Kommentar auszusehen hat, und wo im Pull-Request er platziert werden muss.

Zwei Personen haben sich gewünscht, auf der Startseite des Pull-Requests Kommentare schreiben zu können, welche nicht an eine Position einer Datei gebunden sind. Ein weiterer Wunsch, welcher mehrmals auftrat, forderte die Möglichkeit Zeilenkommentare an Stellen im Code platzieren, an welchen keine Änderungen vorgenommen wurden.

Die Implementierung des ersten Wunsches würde voraussetzen, dass die Nutzer in dem Kommentar auch den Dateipfad und die Codezeile angeben, sodass der Refactoring-Bot die Anweisung mit einem Ort im Quellcode in Verbindung setzen kann. Solche Kommentare zu schreiben wäre jedoch für die Nutzer des Refactoring-Bots aufgrund der Länge und Komplexität sehr unangenehm. Da GitHub nur ermöglicht Zeilenkommentare auf angepasstem Quellcode zu platzieren, kann man den zweiten Wunsch nur mit der Implementierung des ersten Wunsches erfüllen.

### 6.1.3. Unklar gestellte Aufgaben

Die im Kapitel 6.1.2 erwähnten Kommentare, welche nicht unterstützte Refactorings fordern, lassen vermuten, dass einige Testpersonen das beigefügte Aufgabenblatt nicht korrekt verstanden haben. Das Problem sollte mit einer angepassten Version des Aufgabenblattes einfach behoben werden können. Die festgestellten Probleme kann man jedoch einfach auf ein realitätsnäheres Szenario übertragen, in welchem der Bot auf öffentlichen Projekten arbeitet. In solch einem Szenario hat ein Projektverantwortlicher kein Aufgabenblatt und kann prinzipiell jedes Refactoring an jeder möglichen Position im veränderten Quellcode fordern. Da der aktuelle Refactoring-Bot eine stark überschaubare Anzahl an Refactorings unterstützt, könnte das zu frustrierten Nutzern führen, denen kein einziges Refactoring mithilfe des Bot gelingt. Genau dieses Problem wurde in den Interviews einiger Testpersonen genannt, in welchem sie sich gewünscht haben, dass die vom Bot unterstützten Refactorings, in irgendeiner Weise kenntlich gemacht werden.

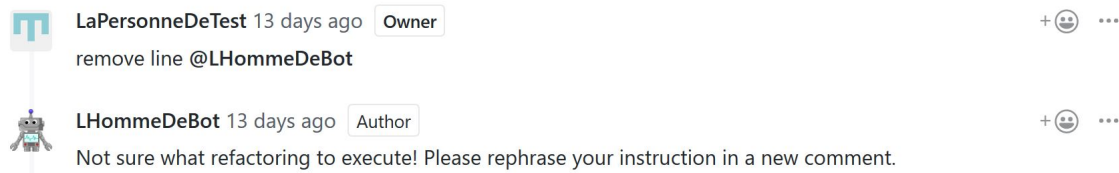
Eine Erweiterung der im Schaubild 3.13 aufgeführten Bedienungsanleitung sollte dieses Problem beheben. So könnte diese mit einer Aufzählung der unterstützten Refactorings erweitert werden. Alternativ könnte auch ein Link zur Refactoring-Bot-Wiki<sup>1</sup> hinterlegt werden, in welcher alle aktuell unterstützten Refactorings beschrieben sind.

---

<sup>1</sup><https://github.com/Refactoring-Bot/Refactoring-Bot/wiki/Pull-Request-Comment-Syntax>

#### 6.1.4. Stellenweise schwaches Feedback

Während das Feedback des Refactoring-Bots generell gut aufgenommen wurde, gibt es ein Szenario, in welchem die Testpersonen mit dem Feedback unglücklich waren. In diesem Szenario erkennt der Refactoring-Bot im Kommentar des Nutzers kein von ihm unterstütztes Refactoring. Im Schaubild 6.1 wird ein solches Szenario veranschaulicht.



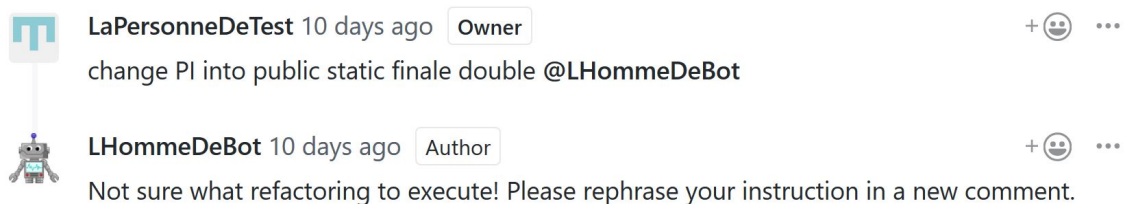
**Abbildung 6.1.:** Nicht verstandener Kommentar

Um das Feedback in solchen Fällen zu verbessern, könnte der Bot seinen Kommentar mit dem Link zu seiner Wiki<sup>2</sup> erweitern, um den Nutzern alle unterstützten Refactorings mit Beispielsätzen vorzuschlagen.

Außerhalb dieses Szenarios scheint das Feedback der Nutzer durchgehend positiv zu sein, was auch dem Schaubild 5.10 entnommen werden kann. Darüber hinaus kann man auch mithilfe des Schaubilds 5.7 feststellen, dass erfolgreich bearbeitete Refactorings im Schnitt nicht mehr als zwei Kommentare benötigen haben, sodass man generell sagen kann, dass das Feedback des Bots dem Nutzer hilft, die auftretenden Probleme in den Griff zu bekommen.

#### 6.1.5. Stellenweise ungenaue Spracherkennung

Wie im Kapitel 5.2 beschrieben wurde, ist der Großteil der Probleme auf eine schwache Spracherkennung zurückzuführen. Diese Probleme treten meistens dann auf, wenn die Kommentare zu viele, oder zu wenige Details enthalten. Ein Beispiel für solch einen Kommentar ist im Schaubild 6.2 dargestellt.

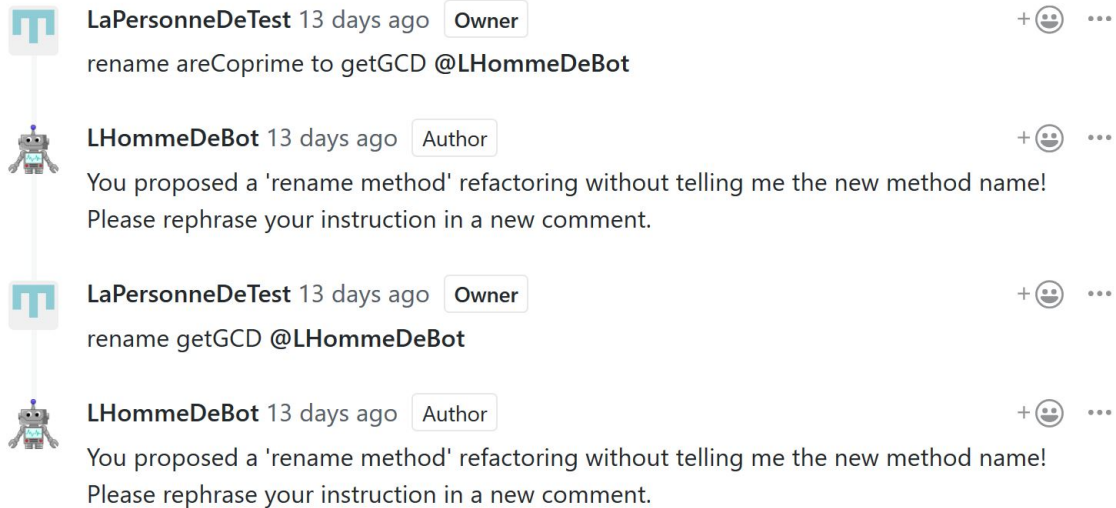


**Abbildung 6.2.:** Sehr detaillierte Anweisung

Um solche Probleme zu lösen, kann auch hier das Feedback um eine Liste von Refactorings mitsamt einiger Beispiele erweitert werden.

<sup>2</sup><https://github.com/Refactoring-Bot/Refactoring-Bot/wiki/Pull-Request-Comment-Syntax>

Neben zu vagen oder komplexen Kommentaren gab es jedoch auch einige präzise Kommentare, welche vom Bot nicht erkannt worden sind. Das Schaubild 6.3 zeigt solch einen Kommentar.



**Abbildung 6.3.:** Beispiel einer schlechten Spracherkennung

Da die Spracherkennung an dieser Stelle versagt hat, schien der Nutzer zu denken, eine noch einfachere Anweisung im Anschluss schreiben zu müssen, welche vom Bot erst recht nicht verstanden wurde.

Solche Kommentare sind nach der Durchführung der Studie in das Training der Wit-Anwendung aufgenommen worden. Sollte man nun Kommentare dieser Art verwenden, wird der Refactoring-Bot diese verstehen.

## 6.2. Limitationen

### 6.2.1. Größe der Stichprobe

Im Rahmen dieser Studie haben sich neun Testpersonen mit dem Refactoring-Bot beschäftigt. Wie im Kapitel 6.1.5 schon erwähnt, wurden unter den Kommentaren der neun Personen schon solche entdeckt, welche vom Bot nicht erkannt worden sind, obwohl sie völlig legitim waren. Dies liegt daran, dass mit einer natürlichen Sprache bestimmte Bedürfnisse auf sehr viele verschiedene Weisen ausgedrückt werden können. Die Wit-Anwendung des Refactoring-Bots ist aktuell nicht in der Lage einen hinreichenden Teil dieser Ausdrucksmöglichkeiten abzudecken.

Damit genügend Testdaten für den Refactoring-Bot und seine Wit-Anwendung gesammelt werden können, muss eine quantitative Studie in einem größeren Umfeld durchgeführt werden. So könnte man den Bot drauf ansetzen, Pull-Requests auf öffentlichen und aktiven Projekten zu eröffnen, um im Anschluss die Kommentare dieser Pull-Requests ins Training der Wit-Anwendung aufzunehmen.

Diese Art von Studie lässt sich aktuell schwer durchführen, da der Refactoring-Bot aktuell nur simple Refactorings einer SonarCloud-Analyse unterstützt. Wie von zwei Personen bemängelt wurde, ändert der Refactoring-Bot in seinem Pull-Request dabei meist nur eine Zeile Code, sodass nicht viele Möglichkeiten entstehen, den geänderten Code im Pull-Request zu kommentieren.

Damit solch eine Studie durchgeführt werden kann, müssen komplexere Refactorings implementiert werden, wie beispielsweise das für diese Studie simulierte „extract method“ Refactoring, welches aktuell im Rahmen einer anderen Arbeit implementiert wird.

Ebenfalls würde sich anbieten, nicht für jedes simple Refactoring einen eigenen Pull-Request zu erstellen, sondern diese in einem Pull-Request zu bündeln. Auf diese Weise würde die Anzahl der geänderten Zeilen auch bei simplen Refactorings steigen, sodass auch mehr Möglichkeiten für ein kommentargesteuertes Refactoring entstehen würden.

### 6.2.2. Kein realistisches Szenario

In einem echten Szenario würde der Refactoring-Bot einem Projektverantwortlichen Änderungen zu seinem eigenen Code vorschlagen. In der durchgeführten Studie haben die Testpersonen mit einem Projekt gearbeitet, welches sie zuvor nie gesehen haben. Das Verständnis über die vom Bot getätigten Änderungen ist also in einem realistischen Szenario deutlich höher.

Des Weiteren wurden den Testpersonen die zu behehenden Code-Smells mithilfe eines Arbeitsblattes kenntlich gemacht. Solch ein Arbeitsblatt würde in einem realistischen Szenario nicht existieren. Deswegen kann es sein, dass ein Projektverantwortlicher keine, oder vom Bot nicht unterstützte Code-Smells entdeckt. Dies könnte zu den selben Problemen führen, welche bei einigen Testpersonen aufgetreten sind, die trotz des Arbeitsblattes den Bot angewiesen haben, nicht unterstützte Refactorings durchzuführen.

Auch wenn die in dieser Arbeit durchgeführte Studie nicht einem realistischen Szenario entspricht, so ermöglicht sie trotz allem die Evaluation der Bedienung und Sprachverarbeitung der aktuellen Implementierung des kommentargesteuerten Refactorings in einem kleineren Rahmen. Da der Refactoring-Bot aktuell nur drei Refactorings mittels der statischen Codeanalyse durchführen kann, und dabei nur minimalistische Änderungen vornimmt, wäre die Durchführung einer realitätsnäheren Studie schwer. Das liegt an den minimalistischen Änderungen die der Bot vornimmt, da in solchen Fällen ein Projektverantwortlicher kaum Möglichkeiten hat im Pull-Request Zeilenkommentare zu platzieren, da diese nur auf veränderten Code geschrieben werden können. Um eine realistischere Studie durchführen zu können, müssen weitere Refactorings implementiert werden. Dabei sollten es solche sein, die mehr Veränderungen am Code vornehmen und bestenfalls eigene Code-Smells mit sich bringen. Ein Beispiel für so ein Refactoring wäre ein „extract method“ Refactoring, welches eine zu lange Methode in zwei unterteilt und die neu entstandene Methode zufällig benennt.

### 6.2.3. Potentielle Manipulation

Die Idee der im Aufgabenblatt enthaltenen Anweisungen war, dass die Testpersonen sich völlig auf die Bewältigung der Interaktion mit dem Bot konzentrieren können, und nicht in einem völlig unbekanntem Projekt nach Code-Smells suchen müssen. Jedoch kann es sein, dass genau diese Anweisungen die Testpersonen manipuliert haben, bestimmte Kommentare so zu formulieren, wie sie es schlussendlich getan haben.

#### **Reordered modifier on field 'PI'**

- Dieser Pull-Request hat noch ein Attribut mit inkorrekt sortierten JAVA-Modifikatoren. Korrigieren Sie das.

#### **Abbildung 6.4.:** Beispiel einer möglichen Manipulation

Das Schaubild 6.4 demonstriert, wie die Anweisung aus dem Aufgabenblatt eine Testperson manipulieren könnte, einen Kommentar unbewusst so zu schreiben, wie der Bot ihn verstehen würde.

Im Vergleich zur Aufgabe aus dem Schaubild 6.4 waren die anderen beiden Aufgaben so gewählt, dass der vom Bot behobene Code-Smell sich von dem zu bearbeitenden unterschieden hat, sodass hier die Wahrscheinlichkeit für eine Manipulation sehr gering ist. Das liegt daran, dass die Aufgabenbeschreibung auf deutsch formuliert ist, und nicht auf englisch. Außerdem wird in der Aufgabenbeschreibung nicht beschrieben wie der vorhandene Code-Smell zu entfernen ist, er wird lediglich beschrieben. Da das Schaubild 5.5 zeigt, dass die im Schaubild 6.4 erwähnte Aufgabe nicht öfter erfolgreich bearbeitet wurde als die anderen beiden, können wir davon ausgehen, dass die Kommentare der Testpersonen höchst unwahrscheinlich durch Manipulation entstanden sind.



## 7. Fazit

Der Refactoring-Bot ist in der Lage Code-Smells einer statischen Codeanalyse eines GitHub-JAVA-Projekts automatisiert zu beheben, und die Änderungen in einem Pull-Request einem Projektbesitzer vorzuschlagen. Bisher hatte dieser keine Möglichkeit, die im Pull-Request verbleibenden Code-Smells automatisiert vom Bot beheben zu lassen, sodass er auf manuelle Anpassung des Pull-Request zurückgreifen musste. Die aktuelle Implementierung des kommentargesteuerten Refactorings zeigt, dass Code-Smells komfortabel behoben werden können, ohne dass der entsprechende Quellcode von einem Projektbesitzer lokal gespeichert, manuell bearbeitet, und veröffentlicht werden muss.

Mittels der durchgeführten, qualitativen Studie hat sich herausgestellt, dass der Refactoring-Bot, trotz einiger Probleme mit der Sprachverarbeitung und Bedienung, gut bei den Testpersonen angekommen ist. Dabei hat die Mehrheit der Testpersonen angegeben, dass sie den Refactoring-Bot in eigenen Projekten verwenden würden. Um die Sprachverarbeitung und die Bedienung des Refactoring-Bots weiter zu verbessern, sind kleinere Anpassungen und eine größere und realitätsnähere Studie nötig, damit der Bot mit einer größeren Datenvielfalt getestet, und entsprechend trainiert werden kann.

### Ausblick

Diese Arbeit hat gezeigt, dass der Refactoring-Bot mit seiner neu implementierten Funktionalität der kommentargesteuerten Refactorings, eine gute Möglichkeit bietet, seinen Nutzern das Refactoren von Software, welche auf GitHub gehostet ist, zu erleichtern.

Im Kapitel 5.1 wurde gezeigt, dass Nutzer von Filehosting-Services nicht nur GitHub, sondern auch andere Services wie GitLab und BitBucket verwenden. Um die im Kapitel 3.4 vorgestellte, plattformunabhängige Implementierung des kommentargesteuerten Refactorings zu demonstrieren, wurde zusätzlich die Unterstützung des Filehosting-Services GitLab<sup>1</sup> zum Refactoring-Bot hinzugefügt. In Zukunft könnte die Unterstützung weiterer Filehosting-Services implementiert werden, sodass der Refactoring-Bot mehr potenzielle Nutzer finden kann.

Während die plattformunabhängige Implementierung des kommentargesteuerten Refactorings den Entwicklern ermöglicht, schnell und einfach die Unterstützung weiterer Filehosting-Services zu implementieren, koppelt sie auch die Funktionalitäten verschiedener Filehosting- und Analysis-Services in bestimmten JAVA-Klassen, was in Zukunft zu einem großen und komplexen Monolithen führen könnte. Sollten in Zukunft weitere Filehosting- und Analysis-Services unterstützt werden, würde es sich anbieten auf eine Microservice-Architektur umzusteigen. Dabei könnte man die

---

<sup>1</sup><https://github.com/Refactoring-Bot/Refactoring-Bot/pull/50>

Funktionalitäten der Filehosting- und Analysis-Services in eigene Microservices schieben. Diese würden dann beispielsweise die Sammlung und Übersetzung der Daten des entsprechenden Filehosters übernehmen.

Damit kommentargesteuerte Refactorings in einem realistischen Szenario vernünftig durchgeführt werden können, müssen weitere, komplexere Refactorings implementiert werden. Das Problem mit der aktuellen Implementierung des Refactoring-Bots ist, dass dieser nur simple Refactorings behebt und für jeden einzelnen einen separaten Pull-Request erstellt. Dieser enthält somit nur minimalistische Änderungen, was die Möglichkeiten für kommentargesteuerte Refactorings minimiert, da in Pull-Requests nur veränderter Code mit Zeilenkommentaren versehen werden kann.

Ebenfalls bietet es sich an, ein automatisiertes Training der Wit-Anwendung zu implementieren. So könnte der Bot nach mehreren nicht verstandenen Kommentaren den Nutzer fragen was genau dieser für ein Refactoring verlangt. Für die Antwort des Nutzers könnte dann eine domänenspezifische ANTLR-Sprache verwendet werden, die vom Bot immer verstanden wird.

Zum Schluss müssen die Probleme behoben werden, die in der limitierten Studie dieser Arbeit aufgetreten sind. Dafür muss die Bedienungsanleitung und das Feedback des Bots so angepasst werden, dass die Nutzer des Bots in allen Situationen wissen, wie sie den Bot bedienen können. Ebenfalls muss die Spracherkennung verbessert werden, indem die Wit-Anwendung des Refactoring-Bots mit mehr Daten trainiert wird. Dafür sollte eine quantitative Studie durchgeführt werden, welche Daten von vielen verschiedenen Entwicklern sammeln kann. So wäre man in der Lage den Refactoring-Bot mit der größtmöglichen Vielfalt von natürlichsprachlichen Anweisungen zu trainieren, um die Probleme der Sprachverarbeitung zu minimieren, welche bei der Durchführung der Studie dieser Arbeit aufgetreten sind.

## Literaturverzeichnis

- [Aug18] S. Augsten. *Was ist ein Parser?* Okt. 2018. URL: <http://archive.is/CXmmg> (besucht am 19. 10. 2018) (zitiert auf S. 24).
- [Dan06] F. Daniel. *Tutorial: Reguläre Ausdrücke*. 2006. URL: <http://archive.is/FNtzs> (besucht am 14. 04. 2019) (zitiert auf S. 25).
- [DF74] D. D. Chamberlin, R. F. Boyce. „SEQUEL: a structured English query language“. In: Mai 1974, S. 249–264. DOI: [10.1145/800296.811515](https://doi.org/10.1145/800296.811515) (zitiert auf S. 24).
- [Gab17] T. Gabriele. *The ANTLR Mega Tutorial*. März 2017. URL: <http://archive.is/BGmsS> (besucht am 06. 04. 2019) (zitiert auf S. 23).
- [Git19a] GitHub. *About pull request reviews*. 2019. URL: <http://archive.is/IV57S> (besucht am 04. 04. 2019) (zitiert auf S. 15).
- [Git19b] GitHub. *About pull requests*. 2019. URL: <http://archive.is/9b8af> (besucht am 03. 04. 2019) (zitiert auf S. 14).
- [Git19c] GitHub. *About repositories*. 2019. URL: <http://archive.is/a2BBk> (besucht am 03. 04. 2019) (zitiert auf S. 13).
- [Git19d] GitHub. *Reviewing proposed changes in a pull request*. 2019. URL: <http://archive.is/Ci9dD> (besucht am 04. 04. 2019) (zitiert auf S. 11, 15).
- [Gmb19] it-agile GmbH. *Refactoring*. 2019. URL: <http://archive.is/HnCVC> (besucht am 04. 04. 2019) (zitiert auf S. 13).
- [JJ12] G. Jäger, R. James. „Formal language theory: refining the Chomsky hierarchy“. In: Bd. 367. 1598. Juli 2012. eprint: <https://royalsocietypublishing.org/doi/pdf/10.1098/rstb.2012.0077>. URL: <https://doi.org/10.1098/rstb.2012.0077> (zitiert auf S. 25).
- [LSZ18] C. Lebeuf, M. Storey, A. Zagalsky. „Software Bots“. In: Bd. 35. 1. Jan. 2018, S. 18–23. DOI: [10.1109/MS.2017.4541027](https://doi.org/10.1109/MS.2017.4541027) (zitiert auf S. 11, 15–17).
- [Mar15] L. Martin. *The Spring Boot Dashboard in STS - Part 1: Local Boot Apps*. 2015. URL: <http://archive.is/Kr4lw> (besucht am 06. 04. 2019) (zitiert auf S. 22).
- [PHF14] T. Parr, S. Harwell, K. Fisher. „Adaptive LL(\*) Parsing: The Power of Dynamic Analysis“. In: *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*. OOPSLA '14. Portland, Oregon, USA: ACM, 2014, S. 579–598. ISBN: 978-1-4503-2585-1. DOI: [10.1145/2660193.2660202](https://doi.org/10.1145/2660193.2660202). URL: <http://doi.acm.org/10.1145/2660193.2660202> (zitiert auf S. 23).
- [Pv18] E. Paikari, A. van der Hoek. „A Framework for Understanding Chatbots and Their Future“. In: *2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. Mai 2018, S. 13–16 (zitiert auf S. 16).

- [RR14] M. M. Rahman, C. K. Roy. „An Insight into the Pull Requests of GitHub“. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, S. 364–367. ISBN: 978-1-4503-2863-0. DOI: [10.1145/2597073.2597121](https://doi.org/10.1145/2597073.2597121). URL: <http://doi.acm.org/10.1145/2597073.2597121> (zitiert auf S. 13).
- [Tur50] A. M. Turing. „I.—COMPUTING MACHINERY AND INTELLIGENCE“. In: Bd. LIX. 236. Okt. 1950, S. 433–460. DOI: [10.1093/mind/LIX.236.433](https://doi.org/10.1093/mind/LIX.236.433). eprint: <http://oup.prod.sis.lan/mind/article-pdf/LIX/236/433/9866119/433.pdf>. URL: <https://doi.org/10.1093/mind/LIX.236.433> (zitiert auf S. 11).
- [WB19] M. Wyrich, J. Bogner. „Towards an Autonomous Bot for Automatic Source Code Refactoring“. In: Mai 2019 (zitiert auf S. 11).
- [WSS+18] M. Wessel, B. M. de Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, M. A. Gerosa. „The Power of Bots: Characterizing and Understanding Bots in OSS Projects“. In: Bd. 2. CSCW. New York, NY, USA: ACM, Nov. 2018, 182:1–182:19. DOI: [10.1145/3274451](https://doi.org/10.1145/3274451). URL: <http://doi.acm.org/10.1145/3274451> (zitiert auf S. 11, 18).



# A. Einverständniserklärung

## Einverständniserklärung

### Beschreibung

Im Rahmen einer Bachelorarbeit werden Sie zur Teilnahme an einer Studie eingeladen, in der das automatisierte Refactoring mithilfe von Kommentaren in GitHub Pull-Requests untersucht wird.

Ihre Aufgabe wird es zunächst sein einige Pull-Requests zu analysieren um sie im Anschluss mittels eines kommentargesteuerten Refactoring-Bots zu bearbeiten. Zum Schluss werden Sie einen Evaluationsbogen ausfüllen und an einem kurzen Interview teilnehmen um über Ihre Erfahrungen mit dem Refactoring-Bot zu berichten.

### Zeitaufwand

Ihre Teilnahme an der Studie wird einmalig 20-30 Minuten dauern.

### Datenerhebung

Sie schreiben GitHub Kommentare, um einen kommentargesteuerten Refactoring-Bot anzuweisen, Refactoring an bestimmten Stellen im Code durchzuführen. Sie beantworten sowohl einige Fragen zu ihrer Nutzungserfahrung mit dem Refactoring-Bot, als auch einige Fragen zu ihrer Person.

Die erfassten Daten dürfen für wissenschaftliche Zwecke in aufbereiteter und anonymisierter Form von Dritten gelesen und wiederverwendet werden.

### Rechte des Teilnehmers

Die Teilnahme an der Studie ist freiwillig. Sie haben das Recht, die Studie jederzeit abbrechen oder gar nicht erst an ihr teilzunehmen. Aus einem Abbruch ergeben sich keine Nachteile für Sie.

### Risiken und Vorteile

Mit der Teilnahme an der Studie entsteht kein Risiko, außer dass Sie mentale und physische Müdigkeit durch die Arbeitsbelastung erfahren. Die Vorteile liegen im Wissen, welches wir aus den gesammelten Daten erhalten.

### Studienleiter

Falls Sie Fragen, Bedenken oder Beschwerden haben, kontaktieren Sie bitte den Studienleiter.

Ich habe die Einverständniserklärung gelesen und verstanden.  
Ich erkläre hiermit meine freiwillige Teilnahme an dieser Studie.

---

Ort, Datum

---

Unterschrift des Studienteilnehmers



## B. Aufgabenblatt

# Studie: Refactoring Bot Aufgabenstellung

Für diese Studie wurden drei Pull-Requests erstellt, welche Änderungen für ein Test-Projekt vorschlagen. Die in den Pull-Requests getätigten Änderungen beheben zwar bestimmte Code-Smells des Test-Projekts, jedoch enthalten sie noch weitere Mängel. Sie haben die Aufgabe die Mängel dieser Änderungen zu beheben.

Hier sind die Pull-Requests mitsamt ihrer Mängel gelistet.

- 1) Reordered modifier on field 'PI'**
  - Dieser Pull-Request hat noch ein Attribut mit inkorrekt sortierten JAVA-Modifikatoren. Korrigieren Sie das.
- 2) Added override annotation on 'getSquareArea'**
  - Dieser Pull-Request hat noch eine Methode mit einem ungenutzten Parameter. Entfernen Sie diesen.
- 3) Extracted method from 'areCoprime'**
  - Dieser Pull-Request hat noch eine Methode mit einem unpassenden Methodennamen. Ändern Sie diesen zu z.B. "getGCD".





## C. Evaluationsbogen

### Refactoring-Bot Erfahrungsbogen

Bitte nehmen Sie sich kurz Zeit um über Ihre Erfahrungen mit dem Refactoring-Bot zu berichten.

#### Feedback zur Nutzung des Refactoring-Bots (Bitte kreuzen Sie immer nur eine Antwort an!)

---

Die Bedienung des Refactoring-Bots wurde verständlich dargestellt.

trifft zu     trifft eher zu     trifft eher nicht zu     trifft nicht zu

---

Der Bedienung des Refactoring-Bots war intuitiv.

trifft zu     trifft eher zu     trifft eher nicht zu     trifft nicht zu

---

Der Refactoring-Bot hat die Änderungsvorschläge meiner Kommentare korrekt verstanden.

immer     häufig     selten     nie

---

Der Refactoring-Bot hat verstandene Refactoring-Befehle korrekt umgesetzt.

immer     häufig     selten     nie

---

Bei Fehlern oder Problemen gab mir der Refactoring-Bot verständliches Feedback.

trifft zu     trifft eher zu     trifft eher nicht zu     trifft nicht zu

---

Ich würde den Refactoring-Bot in meinem Projekt nutzen wollen.

definitiv     wahrscheinlich     unwahrscheinlich     definitiv nicht

---

---

**Persönliche Informationen (Bitte kreuzen Sie immer nur eine Antwort an!)**

Studiengang: \_\_\_\_\_ Fachsemester: \_\_\_\_\_

Geschlecht:  männlich  weiblich  sonstige

Ich habe Erfahrung mit der Programmiersprache JAVA.

viel  mäßig  wenig  keine

Ich arbeite regelmäßig mit Filehosting-Plattformen wie GitHub.

trifft zu  trifft eher zu  trifft eher nicht zu  trifft nicht zu

Ich arbeite mit diesen Filehostern. (Hier können Sie auch **mehrere** oder **keine** Antworten ankreuzen!)

GitHub  GitLab  BitBucket  andere: \_\_\_\_\_

Ich habe Erfahrung mit Pull-Requests und Pull-Request-Reviews.

trifft zu  trifft eher zu  trifft eher nicht zu  trifft nicht zu

Ich halte das regelmäßige Refactoring von Software für...

sehr wichtig  wichtig  eher unwichtig  unwichtig

---



---

### **Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift